1-1-2015

Evaluating the Robustness of Resource Allocations Obtained through Performance Modeling with Stochastic Process Algebra

Srishti Srivastava

Evaluating the robustness of resource allocations obtained through performance modeling with stochastic process algebra

By

Srishti Srivastava

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2015

Evaluating the robustness of resource allocations obtained through

performance modeling with stochastic process algebra

By

Srishti Srivastava

Approved:

Ioana Banicescu
(Major Professor)

Sherif Abdelwahed
(Committee Member)

Edward A. Luke
(Committee Member)

Edward B. Allen
(Committee Member)

Changhe Yuan
(Committee Member)

T.J. Jankun-Kelly
(Graduate Coordinator)

Jason M. Keith
Interim Dean
Bagley College of Engineering

Name: Srishti Srivastava

Date of Degree: May 08, 2015

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Ioana Banicescu

Title of Study: Evaluating the robustness of resource allocations obtained through performance modeling with stochastic process algebra

Pages of Study: 174

Candidate for Degree of Doctor of Philosophy

Recent developments in the field of parallel and distributed computing has led to a proliferation of solving large and computationally intensive mathematical, science, or engineering problems, that consist of several parallelizable parts and several non-parallelizable (sequential) parts. In a parallel and distributed computing environment, the performance goal is to optimize the execution of parallelizable parts of an application on concurrent processors. This requires efficient application scheduling and resource allocation for mapping applications to a set of suitable parallel processors such that the overall performance goal is achieved. However, such computational environments are often prone to unpredictable variations in application (problem and algorithm) and system characteristics. Therefore, a robustness study is required to guarantee a desired level of performance. Given an initial workload, a mapping of applications to resources is considered to be robust if that mapping optimizes execution performance and guarantees a desired level of performance in the presence of unpredictable perturbations at runtime.

In this research, a stochastic process algebra, Performance Evaluation Process Algebra (PEPA), is used for obtaining resource allocations via a numerical analysis of performance modeling of the parallel execution of applications on parallel computing resources. The PEPA performance model is translated into an underlying mathematical Markov chain model for obtaining performance measures. Further, a robustness analysis of the allocation techniques is performed for finding a robust mapping from a set of initial mapping schemes. The numerical analysis of the performance models have confirmed similarity with the simulation results of earlier research available in existing literature. When compared to direct experiments and simulations, numerical models and the corresponding analyses are easier to reproduce, do not incur any setup or installation costs, do not impose any prerequisites for learning a simulation framework, and are not limited by the complexity of the underlying infrastructure or simulation libraries.

Key words: performance modeling, performance evaluation, robustness analysis, parallel computing, process algebra, stochastic computing environment

DEDICATION

To my beloved father, mother, brother, and husband, who have all been my eternal inspiration, emotional support, and my pillars of strength.

ACKNOWLEDGEMENTS

Foremost, I would like to acknowledge and convey my sincerest gratitude to my major professor and mentor, Dr. Ioana Banicescu, who has continuously guided me towards achieving quality research work, has provided me the required platform, constant encouragement, and unending valuable support to grow as an academic professional, and has helped me in many different ways throughout the entire course of this dissertation. As my major adviser, she led me to a number of research collaborations that have helped increase the qualitative value of my Ph.D. dissertation and has helped my research gain recognition at both national and international levels. I am also very thankful and honored to be considered for nomination by Prof. Banicescu for a number of distinguished academic awards. Her efforts and support allowed me to attend prestigious forums, such as the Heidelberg Laureate Forum. In addition, as a director of the National Science Foundation Center for Cloud and Autonomic Computing at Mississippi State University (NSFCAC at MSU), Prof. Banicescu provided generous financial support and research opportunities. The research work in this dissertation was supported in part by the National Science Foundation (NSF) under grant numbers NSF IIP-1127978 and NSF IIP-1034897 at the NSF Center for Cloud and Autonomic Computing, Mississippi State University and the NSF CORBI grant NSF IIP-1331282.

I am also very thankful to all the members of my graduate committee, for their invaluable advice towards perfecting the research in my Ph.D. dissertation. As a professor in the field of high performance computing, Dr. Luke provided very useful advice towards maintaining the correctness of my research. As a graduate coordinator, Dr. Allen provided me with helpful suggestions and comments for improving my dissertation and for abiding by the required regulations for a timely completion of my Ph.D. studies. As a co-PI on the project for NSFCAC at MSU, Dr. Abdelwahed helped in providing financial support and research opportunities that helped in improving the breadth of my dissertation research. Dr. Yuan provided valuable advice and resources in the field of machine learning that led to a research collaboration on advancing the benefits of the work on robustness of dynamic scheduling with the help of machine learning techniques.

I would like to acknowledge Dr. Reese, department head, for her invaluable support and words of encouragement throughout the course of my graduate studies. In addition, I thank all the faculty members at the Department of Computer Science and Engineering who have been equally supportive and have added value to my knowledge domain in the field of computer science via their graduate courses, teaching, and research. I would also like to acknowledge the support for infrastructure and resources provided by the high performance computing collaboratory at MSU. A special appreciation goes to my colleagues and collaborators, Dr. Ciorba, Nitin Sukhija, Mahadevan Balasubramaniam, Rajat Mehrotra, Dr. Malone, Timothy Hansen, Dr. H.J. Siegel, and Dr. Anthony Maciejewski, for their valuable contributions towards improving the quality of my research work. I would also

like to thank the staff members at the Department of Computer Science and Engineering, who were always willing to help me with everything required during my graduate studies.

On a personal note, I would like to thank my family for their love, sacrifice, and support, at every step of my Ph.D. education. I would like to acknowledge my father, Dr. S.C. Srivastava, for being my first and very strong inspiration towards becoming a quality academic professional. I would like to thank my mother, Mrs. Beena Srivastava, for always being my emotional support and for providing me with her love, care, and blessings. I am deeply grateful to my husband, Dr. Prashant Kumar, for being my pillar of strength and hope, and for his unending love, support, and words of encouragement that have kept me moving towards achieving a high quality Ph.D. I would also like to thank my brother, Srijan Srivastava, for always believing in me and for his continuous love and best wishes for my successful endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Recent developments in the field of parallel and distributed computing has led to a proliferation of solving large and computationally intensive mathematical, science, or engineering problems, that consist of several parallelizable parts and several non-parallelizable (sequential) parts. Amdahl's law states that, "*a small portion of the program which cannot be parallelized will limit the overall speed-up available from parallelization*" [9]. In addition, Gustafson's law states that, "*if you apply P processors to a task that has serial fraction f, scaling the task to take the same amount of time as before, the speedup* = $f + P(1 - f) = P - f(P - 1)$. *It shows more generally that the serial fraction does not theoretically limit parallel speed enhancement, if the problem or workload scales in its parallel component.*" [54]. Therefore, in a parallel and distributed computing environment, the performance goal is to optimize the execution of parallelizable parts of an application on concurrent processors. This requires efficient application scheduling and resource allocation for mapping applications to a set of suitable parallel processors such that the overall performance goal is achieved. However, such computational environments are often prone to unpredictable variations in application (problem and algorithm) and system characteristics. Therefore, a robustness study, of resource allocation and schedul-

ing, is required to guarantee a desired level of performance. *Given an initial workload, a mapping of applications to resources is considered to be robust if that mapping optimizes execution performance and guarantees a desired level of performance in the presence of unpredictable perturbations, in application and system characteristics, at runtime*. The knowledge of a robust mapping is useful in designing an efficient resource allocation system that can maintain a desired level of execution performance in the presence of tolerable variations in application and system parameters.

1.1.1 Performance Evaluation

The rapid development of computing technology has increased the complexity of computational systems and the ability to solve large, more complex problems. However, the need for system developers and users to measure the performance of these systems has been constant throughout this proliferation. Researchers and scientists from various fields are interested in accurate modeling and simulation of various complex phenomena from various scientific and enterprise areas. These simulations are often routines that perform tasks as repetitive computations over very large data sets. Moreover, their nature (or computational requirements) may be irregular, rendering one task likely to take more time than other tasks, depending on the application. The resources in a large-scale system are widely distributed and highly heterogeneous, often shared among multiple users, and their availability cannot always be guaranteed or predicted. Hence, the quality and quantity of available resources to a single user are continuously changing. Running computationally intensive applications in heterogeneous environments exhibits irregular behavior, in general due to variations in pa-

rameters of problem, algorithm and system. *Performance evaluation* is concerned with the description, analysis and optimization of such dynamic behavior of parallel and distributed systems. The goal of performance evaluation is to understand the behavior of the system (which includes the application and the computational system) and identify the aspects of the system that are sensitive from a performance point of view. In general, a performance study addresses an objective, which is achieved via evaluating several alternative solutions to the intended problem, and often requires solution to a multi-objective optimization problem of a utility function ($U = f(makespan, robustness, power, cost, andothers)$). In parallel and distributed computing, resource management is an important and active research area, which is often challenged by the problem of finding a mapping of tasks to machines that optimizes a system performance feature while maintaining an acceptable level of quality of service. The work done in this research is focused on evaluating resource allocations in a dynamic environment which is prone to unpredictable variations in application and system characteristics by analyzing the effect of probable runtime perturbations on the execution performance obtained as a result of using a certain resource allocation.

1.1.2 Robustness Analysis

Scheduling applications on large-scale platforms, where chances of workload variation and faults are high, require an approach to ensure the robustness of the underlying mapping. In earlier work on robustness of resource allocations/task scheduling algorithms, robustness was addressed individually for a single method, or even for a single application [107] and has been discussed in the following chapter. As robustness has various

definitions under different contexts and applications, it has not yet been possible to give it a universally valid definition for all circumstances. Moreover, designing a scheduling algorithm with the goal of achieving robustness gives no guarantee that the algorithm is more robust than an algorithm designed without that goal. It is a fact that today's high-performance computing systems have rapidly evolved in size (from multi-core to many-core and to petascale), increased in complexity of the interconnection networks and of the processor hierarchies, and consequently, have become more expensive in terms of energy consumption and performance per watt. Nowadays, time to solution consists of more factors than just the execution time of the application [45]. It naturally follows that new metrics are needed to characterize the application performance, in addition to the traditional performance metrics, such as execution time, efficiency, scalability, and others. The new metrics must characterize the robustness of scientific applications running on the complex high-performance computing systems. Therefore, a study of robustness is essential to ensure the level of performance of the techniques used to parallelize scientific applications under highly unpredictable conditions, and to develop metrics that can measure the robustness of the scheduling techniques with respect to various causes that degrade performance. Although much work has been focused on formulating robustness metrics for a number of resource allocation techniques [106] and a number of dynamic loop scheduling (DLS) techniques [15][107], developing formal analytical models for evaluating the robustness of such techniques on parallel and distributed systems is still an open problem.

Analytical modeling of robustness can provide a platform for predicting the robustness of various *application-to-machine* allocations and thereupon for selecting the most robust

allocation for a parallel execution of scientific applications on systems that are often prone to uncertain variations in their computational environmental factors. The proposed analytical modeling of robustness can provide a platform for predicting the performance of executing applications on parallel machines, such as makespan, throughput, and resource utilization [59]. Further, the modeling can also be extended to include other metrics, such as robustness, power consumption, and others, to ensure an efficient, robust, and power effective execution of scientific applications on parallel and distributed computing systems, via modeling a single utility function that includes all the performance features of interest as mentioned above.

1.1.3 Process Algebra for Performance Evaluation

The approaches for performance evaluation of computer and communication systems have been broadly classified into three types: analytical and numerical modeling, simulation, and direct experiments [70]. The choice of a performance evaluation technique is dependent upon many key factors such as, the stage of the system under evaluation (for example, if a system already exists and requires postmortem analysis or if a new system needs to be created and requires a predictive analysis), need for generality, analysis time, tools required, comparative analysis of systems, and cost. In parallel and distributed computing systems, direct experimentation requires the availability of a system in which all the parameters can be controlled. Such an approach is costly, time consuming, difficult to replicate (for a comparative analysis), and lacks generality. Simulations provide a more cost-efficient approach towards performance evaluation of parallel and distributed comput-

ing systems. An abstraction of the system is simulated using specific simulation languages, tools, and techniques. However, developing a simulation platform to incorporate all of the essential system properties is a time consuming process, it compromises accuracy and often incurs an intellectual burden of validating the simulations and evaluating the correctness of the simulated system via calculation of confidence intervals. In contrast, analytical and numerical modeling for performance evaluation allows derivation of an expression of the performance feature of interest in terms of the input parameters of the model. In case of a predictive analysis of a computing system, analytical models generally provide the best insight into the effects of various parameters and their interactions, and are easier to replicate for a comparative analysis of different systems. In addition, analytical and numerical modeling provides the most cost efficient approach towards performance evaluation of parallel and distributed computing systems [19][21][59].

Markovian models have been shown to be an effective technique for performance analysis of computer and communication systems, where the system components are modeled as Markov processes and the overall performance (for example, throughput, resource utilization, and others) is evaluated upon the numerical analysis of these Markov processes [114]. However, construction of Markov processes is a tedious task for large parallel and distributed systems. Therefore, a intermediate system description language is often used to model and design the system components and their behavior. Process algebras are abstract languages used for specification and design for parallel and distributed systems. The motivations for investigating the use of process algebras for performance modeling can be regarded as arising from three distinct problems of performance analysis which

have been identified in the recent years and are listed as follows along with a description of how the adoption of process algebra as a performance modeling language has implications for each of these problems [59][60]:

- *Integrating performance analysis into system design*: using a formal description language for performance modeling such as process algebra allows a closer integration of performance analysis into design methodologies. This enables a timely consideration of performance aspects for designing a parallel and distributed computing system.
- *Representing systems as models*: modern parallel and distributed systems do not fit into the traditional models of sequential control and resource allocation. The components within modern parallel and distributed computing systems work in a framework of autonomy and cooperation. A process algebra model consists of a system of components known as active agents who are defined by their individual behaviors and these agents interact via the cooperator paradigm of the process algebra.
- *Model tractability*: process algebra models offer techniques such as model simplification and aggregation that enable performance analysis of very large systems. The compositionally of process algebra allows a system designer to evaluate a part of the model while maintaining the integrity of the overall model.

In this work, Performance Evaluation Process Algebra (PEPA) [59] is used for performance modeling of resource allocations for mapping scientific applications to parallel machines, followed by an analysis of the robustness of the resource allocations. The motivation behind using PEPA, for performance modeling and evaluation of resource allocations in parallel and distributed computing systems, is the theoretical development of PEPA that captures the advantages of analytical modeling via process algebra and derivation of performance measures via the underlying mathematical structure. A more detailed theoretical description of process algebra and PEPA is given in Chapter 2.

1.2 Thesis Statement

Hypothesis: the numerical models of resource allocations (for mapping applications to parallel machines) obtained using the performance evaluation from stochastic process algebra can be successfully used for obtaining a robust solution to the mapping problem in parallel computing systems.

The focus of this research is a study to address the challenging issues concerning the evaluation of robustness of initial mappings used for scheduling applications on today's emerging parallel computing systems (clusters, grid environments, clouds, and others). This approach is inclined towards evaluating robustness of the performance of the resource allocations modeled numerically using a stochastic process algebra. The goal is to develop analytical and numerical models of resource allocations for parallel execution of applications, that have varying workload, on parallel and heterogeneous computing resources. The underlying theoretical foundation is constructed based on the established quantitative concepts of the stochastic process algebra (SPA). Further, the benefits of the proposed analytical model of robustness are discussed highlighting its utility towards a holistic approach that can ensure a robust execution of applications on modern and future parallel computing systems (including autonomic computing systems (ACS) and cloud computing systems) by selecting the most robust mapping obtained from the analysis of the analytical model.

Novel contributions as well as the contributions that led to the research in this dissertation are listed below.

1. Performance modeling of resource allocations in parallel and distributed computing using PEPA.

2. Robustness evaluation using a passage time analysis (numerical analysis of Markovian models) of the developed performance models of resource allocations.

3. Robustness analysis of resource allocations w.r.t. equal variation in workload across all applications, and validating the robustness results of resource allocations obtained from performance modeling using PEPA with the robustness of the same resource allocations obtained via prior simulation experiments.

4. Robustness analysis of resource allocations w.r.t. the non-uniform variation in workload across all applications.

5. First implementation of the DLS methods in a simulation framework for a comparative analysis of the execution performance of these methods [108].

6. Study of the use of a model free machine learning (reinforcement learning) approach towards an automatic selection of the best DLS method for scheduling time-stepping scientific applications [16].

7. Formulation of robustness metrics for dynamic scheduling methods used in parallel computing systems and a study towards an online selection, of the most robust dynamic scheduling method, using machine learning techniques. The study of robustness of dynamic scheduling is applicable to a class of time-stepping scientific applications and is a part of the foundation work on robustness that has led to this research [107][110].

8. First implementation of a learning based methodology for an online prediction of the robustness of DLS methods using an artificial neural network [109].

9. A power-aware execution of scientific applications parallel and distributed computing systems using an existing model-based framework that combines the functionalities of DLS methods with a feedback limited look ahead controller [87].

10. A combined dual-stage framework for robust scheduling of scientific applications in heterogeneous environments with uncertain processor availability [32].

11. A list of publications, which have resulted from this research work, is given below.

    (a) I. Banicescu, F. M. Ciorba, and S. Srivastava, "Chapter 22: Performance Optimization of Scientific Applications using an Autonomic Computing Approach", pp. 437466, in Scalable Computing and Communications: Theory and Practice, 2013, John Wiley & Sons, Inc.

    (b) N. Sukhija, B. Malone, S. Srivastava, I. Banicescu, F. M. Ciorba. "Portfolio-based Selection of Robust Dynamic Loop Scheduling Algorithms Using Machine Learning," In Proceedings of the 15th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS-ParLearning) 2014, Phoenix, AZ, USA, On CD-ROM, IEEE Computer Society Press.

(c) S. Srivastava, B. Malone, N. Sukhija, I. Banicescu, F. M. Ciorba, "Predicting the Flexibility of Dynamic Loop Scheduling Using an Artificial Neural Network," In Proceedings of the IEEE International Symposium on Parallel and Distributed Computing (ISPDC2013), Bucharest, Romania. On CD-ROM, IEEE Computer Society Press.

(d) N. Sukhija, I. Banicescu, S. Srivastava, F. M. Ciorba, "Evaluating the Flexibility of Dynamic Loop Scheduling on Heterogeneous Systems in the Presence of Fluctuating Load using SimGrid," In Proceedings of the 14th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS-PDSEC) 2013, Boston, USA, On CD-ROM, IEEE Computer Society Press

(e) S. Srivastava, N. Sukhija, I. Banicescu, F.M. Ciorba, "Analyzing the Robustness of Dynamic Loop Scheduling for Heterogeneous Computing Systems," In proceedings of the 11th IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2012)

(f) F. M. Ciorba, T. Hansen, S. Srivastava, I. Banicescu, A. M. Maciejewski, H. J. Siegel. "A Combined Dual-stage Framework for Robust Scheduling of Scientific Applications in Heterogeneous Environments with Uncertain Availability". In Proceedings of the 13th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS-HCW) 2012, Shanghai, China: On CD-ROM, IEEE Computer Society Press.

(g) S. Srivastava, I. Banicescu, F.M. Ciorba, W.E. Nagel, "Enhancing the Functionality of a GridSim-Based Scheduler for Effective Use with Large-Scale Scientific Applications," in Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2011), vol., no., pp.86-93, 6-8 July 2011.

(h) S. Srivastava, I. Banicescu, F.M. Ciorba, "Investigating the robustness of adaptive Dynamic Loop Scheduling on heterogeneous computing systems," in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, vol., no., pp.1-8, 19-23 April 2010.

(i) S. Srivastava, F. M. Ciorba, I. Banicescu, "Employing a Study of the Deterministic Robustness Metrics to Assess the Reliability of Dynamic Loop Scheduling," in Proceedings of the IEEE High Performance Computing Conference (HiPC 2010) - Student Research Symposium (SRS), Goa, India:, IEEE Computer Society Press.

CHAPTER 2

BACKGROUND

With the advent of increasingly complex computation and communication systems that have evolved from a single node machine to parallel and distributed computing systems containing clusters of very powerful (multiprocessor, muti-core, and others) machines, there is a need for a robust design of these computing and communication systems. In general, the textitmapping problem, which is scheduling independent tasks (or applications) onto a set of heterogeneous parallel processors, is known to be NP-Complete [34][44][68]. This research is a step towards the ongoing efforts towards achieving a robust schedule for allocation of independent tasks on heterogeneous machines in parallel and distributed computing systems.

2.1 Robustness

In advanced computing systems, robustness is a quality which defines their ability to withstand changes in procedures or their working environments. Such systems are defined robust if they are capable of coping with unpredictable variations with minimal damage in their functionality. As robustness has various definitions under different contexts and applications, it has not yet been possible to give it a universally valid definition for all circumstances. The IEEE standard glossary defines robustness as "the degree to which a

system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" [77]. Invalid inputs include errors above tolerance levels, such as faults and in this context, robustness might be interpreted as the degree of the system's ability to handle exceptions, such as to tolerate faults. Most of the current research in this area concerns the design of methods that address issues of fault-tolerance and resilience.

2.1.1 Robustness of Static Resource Allocation

The initial work on robust scheduling originated from job-shop application scheduling frameworks. Robustness measures and robust scheduling methods have been developed to schedule job-shop applications [82]. The authors define schedule robustness as a measure of the post disturbance makespan and the post disturbance makespan variability. The authors utilize a right-shift control policy when a job is interrupted in the presence of a disturbance to maintain the schedule performance. The impact of a disturbance on the job-shop performance is minimized via employing a robust initial schedule, where the robustness is defined as a weighted function of the expected makespan and the expected delay. Standard branch and bound approach is used to solve the NP-Hard robust scheduling problem (RSP) to obtain a robust schedule for N independent jobs on a single machine [35]. A robust schedule is obtained for scheduling metaprograms on a computational grid [26]. The authors implement a *naive model using a proposed empirical formula* to calculate the robustness of a schedule. During the process, a current schedule is replaced with another schedule yielding a smaller execution time and all the schedules yielding a larger execution time than the current schedule are rejected. Work has also been done on

improving the robustness or flexibility of a schedule in a job-shop execution environment, by minimizing the lateness factor and by using neighborhood-based methods for recovering from a disturbance [72]. A stochastic mixed integer programming (SMIP) approach is used to obtain a robust resource allocation for parallel and distributed systems [49]. The linearization of the mixed integer programming (MIP) formulation of the resource allocation mapping problem leads to a robust initial resource allocation, and the robustness of the mapping is measured as the amount of additional workload that the system can handle at runtime while maintaining the performance. As a subsequent work, the authors generate a more robust resource allocation using iterative integer programming (IIP) [50]. The resource allocation obtained upon the IIP formulation of the mapping problem allows some slackness (δ), which is provided by the user of the application. The authors involved with the INRIA GRAAL project, propose a fault tolerant scheduling algorithm (FTSA), based on an active replication scheme. The authors identify the reliability of a scheduling algorithm as a guarantee of application performance in the presence of processor failures on heterogeneous computing platforms [22].

Designing a scheduling algorithm with the goal of achieving robustness gives no guarantee that the algorithm is more robust than an algorithm designed without that goal. In such a situation formulating robustness metrics can be helpful to measure the robustness of a scheduling method against possible erroneous inputs or variable environmental factors. A general methodology for developing robustness metrics for resource allocation has been presented by Ali et al. [6]. The authors present state of the art work for formulating a robustness metric for finding the most robust initial resource allocation for

heterogeneous systems prone to uncertain perturbations, such as, unexpected system load variations, processor failures, and others. They provide a mathematical description for the robustness metric followed by a procedure for deriving the robustness metric. The proposed procedure, to formulate the generalized robustness metric, is termed as a FePIA (Feature Perturbation Impact Analysis) procedure. For resource allocations, the authors give a customized definition as follows: "a resource allocation is defined to be robust with respect to specific system performance features against perturbations (uncertainties) in specified system parameters if degradation in these features is constrained when limited perturbations occur" [6]. Further, the authors also address three questions on robustness and render them as mandatory for claiming robustness of resource allocations in heterogeneous parallel computing systems [5]. The three questions are: (i) what behavior of the system makes it robust? (ii) What uncertainties is the system robust against? And (iii) quantitatively, how robust is the system? In their initial work, the authors formulate a deterministic robustness metric (DRM) that uses a scalar estimate of the execution time of each application on each machine to determine the robustness of a resource allocation [6][5][99][7]. The DRM is obtained via employing the FePIA procedure and is defined as the variation in the perturbation parameter that can be tolerated by the system before the performance feature of interest exceeds an allowable range. The goal is to obtain a resource allocation that can yield the largest value of the DRM. This is further formulated as a convex optimization problem, where the goal is to maximize the value of the DRM and the constraints are given by the allowable range of values for the performance feature of interest. The authors use a number of greedy heuristics that use the DRM to solve

the optimization problem for obtaining the most robust resource allocation [6][5][99][7]. In their subsequent work, a stochastic robustness metric (SRM) is proposed to determine a robust static resource allocation via a mathematical model that describes the stochastic relation between uncertainty in system parameters and its impact on system performance [101][104][103][100][111][71][90][98][4][102]. The SRM is defined as the probability that a performance feature is confined in an allowable interval in the presence of perturbations and does not exceed the QoS constraints (such as deadline). The SRM uses information about the execution time distributions of the application for the robustness of the resource allocation. Thus, the resource allocation obtained upon employing the SRM is associated with a probability. The goal is to obtain a resource allocation that provides the maximum value for the SRM. This goal is formulated as a solution to an optimization problem of maximizing the SRM value constrained by the QoS attributes desired from the application execution. A number of heuristic techniques are utilized to find an optimized resource allocation that can provide the highest value for the SRM. The authors use heuristics based on greedy search algorithms that optimize the SRM to find a robust resource allocation in a stochastic computing environment [104][71]. A number of iterative search algorithms, such as ant colony optimization, steady state genetic algorithms, and simulated annealing, have been used in [100][71]. Immediate mode heuristics and batch mode heuristics have also been utilized to solve the optimization problem in [90]. To solve the optimization problem under memory constrained systems, the authors also experimented with branch and bound heuristics based on integer linear programming [98]. A comparison between the set of heuristics (greedy, genetic algorithms, simulated annealing, and ant

colony optimization) that were utilized for obtaining a solution to the optimization problem for maximizing the SRM, has been presented in [4][102]. A more comprehensive description of robustness of resource allocations in heterogeneous systems, deterministic models of robustness using a DRM, stochastic models of robustness using an SRM, and a comparison between the DRM and SRM has been discussed in [106][102].

2.1.2 Robustness of Dynamic Scheduling

The study of robustness of scheduling scientific applications on high-performance parallel and distributed computing systems is a two-faceted issue that can be addressed at the system level and at the application level. Therefore, it is of interest to investigate the robustness of scheduling techniques at the application level, which together with studies conducted at the system level constitute a holistic approach to ensure robustness of executing the scientific applications on modern and future computing systems [42]. To address this issue, in addition to the work done towards the evaluation of robustness of static resource allocation [6][5][99][7][101][104][103][100][111][71][90][98][4][102], research studies have also been conducted towards analyzing the robustness of a number of DLS methods, for dynamic scheduling of scientific applications on large-scale parallel and distributed system of heterogeneous processors, in the presence of varying processor loads (defined by the flexibility metric) and processor failures (defined by the resilience metric) [15][107]. A mathematical formulation of the *flexibility metric* (used for measuring the flexibility of DLS algorithms in the presence of fluctuating processor availabilities that results in a variation of the load on that processor) and the *resilience metric* (used for mea-

suring the resilience of DLS algorithms in the presence of processor failures), using the FePIA procedure as described in [6][5], has been presented as preliminary work towards analyzing the flexibility and resilience of the DLS algorithms for dynamic scheduling of computationally intensive, irregular, and data parallel scientific applications on parallel and distributed computing systems that are prone to runtime variations in the problem, algorithm, and system characteristics [15][107]. In subsequent work, simulation, of the dynamic scheduling (using DLS algorithms) and the execution of the scientific applications on a set of parallel and heterogeneous processors, has been performed to compare the load balancing characteristics of the DLS methods, and to evaluate the flexibility of the DLS methods in the simulated parallel and distributed computing environment [108][110][112]. In earlier work, the performance evaluation of the DLS methods has been obtained experimentally, using real world applications executing on real physical machines [13][27]. However, conducting the real experiments was a tedious and time-consuming task. In addition, the results obtained from these real experiments were difficult to reproduce for a fair comparison of the DLS methods, given the same computing environments. To overcome this limitation, a simulation of scheduling scientific applications on a cluster of grid resources via the DLS techniques, was presented as the first step towards evaluating the flexibility of the DLS algorithms [108]. The simulation framework was provided by *Alea* [76], which is a *Gridsim* [25] based simulator specifically designed for task scheduling on parallel and distributed computing systems. Alea provided a platform to simulate the execution of different types of irregular loop iteration execution times (representing variations in application characteristics), on different sizes of computing systems (representing hetero-

geneity among processors) with the required level of detail to assess the performance (in terms of the parallel execution time), and thereafter the flexibility the DLS methods. The basic structure of the Alea simulator, used for the simulation of scheduling loop iterations modeled as individual tasks of a scientific application on a grid environment, is illustrated in Figure 2.1.

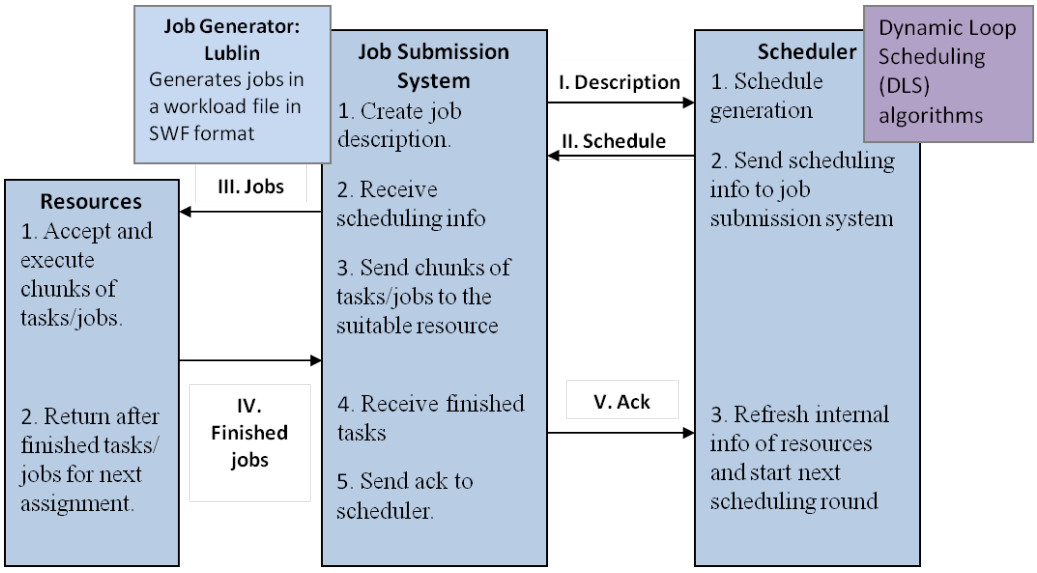

Figure 2.1: Architecture of Alea's functionality extended with DLS algorithms

Further, a simulation, using priority queues written in the C language, has been used to simulate the execution of data parallel loops within a scientific application exhibiting irregular behavior, and to simulate the uncertainty in the variation of processor weights using random probability distributions [110]. The simulation results are used to experimentally analyze and evaluate the robustness of the DLS methods for various execution scenarios, which reflect the performance of the DLS methods in both dedicated and non-dedicated

computing environments. To exemplify the execution environments, a set of tolerable threshold values is chosen, and the flexibility of the DLS methods is measured (using the FePIA procedure as described in [6][5]) against the variation of the computational speed of the processors, with respect to the chosen threshold values.

A flexibility analysis of DLS techniques against fluctuating system load at a larger scale by employing the experimental scenarios of different combinations of problem size, computing systems size, and scheduling methods has been conducted using SimGrid [28], which is a more complex simulator as compared to the C-based simulation using priority queues used in [110]. The robustness analysis of the DLS techniques has been performed considering the fluctuation of the system load as a compound effect of both the algorithmic and the systemic variances, which are modeled using probability distributions. The choice of various distributions representing variances in applications and system characteristics has been made for the purpose of identifying the effects of the resulting irregularities on the DLS techniques, thus leading towards finding a measure of their flexibility. The use of the Simgrid simulation toolkit is motivated by the need to overcome the constraints involved in testing the performance of the DLS methods repeatedly on real systems because real testbeds are time intensive to create, have limited control over the dynamic behavior of the computing system, and it is often difficult to perform repeated and controlled experiments to schedule scientific applications composed of a large number of computationally intensive loop iterations on real computing systems.

2.1.3 A Combined Dual-stage Framework for Robust Scheduling of Applications in Stochastic Computing Environments

Scheduling parallel applications on existing or emerging computing platforms is challenging, and, among other attributes, must be efficient and robust. The need for robustness at both, the system and the application, levels motivated the study of a dual-stage framework evaluate the robustness of efficient resource allocation and dynamic load balancing of scientific applications in heterogeneous computing environments with uncertain availability [32]. The first stage employs heuristics that produce a robust resource allocation for an initial static mapping of applications to machines, while the second stage incorporates robust dynamic loop scheduling techniques for a robust scheduling and execution, of those applications on the allocated machines, at runtime. The combined dual-stage framework constitutes a comprehensive framework that enables and provides guarantees for the robust execution of scientific applications in computing systems where uncertainty is caused by various unpredictable perturbations. The research reports on studies for determining the best techniques to be used for each stage that: (a) maximize the probability that the system makespan satisfies a deadline, and (b) minimize the system makespan for every given availability level in the system [32]. The goal of the combined dual-stage framework is to assign applications to heterogeneous computing systems and execute them in such a way that all applications complete before a common deadline, and their completion times are robust against uncertainty in input data and system availability. To accomplish this goal, the approach behind the combined dual-stage framework is to divide the execution of scientific applications on heterogeneous computing systems into two stages, as outlined in

Figure 3.7. In Stage I *initial mapping*, resources are allocated to each application according to a given robust RA policy. Initial mapping (IM) can be defined as the problem of finding a mapping of a batch of applications onto a set of resources to maximize robustness against uncertain input data and system availability. Robustness here is defined as the probability that applications are completed on the allocated resources by a common deadline [102]. In Stage II *runtime application scheduling*, the execution of each application is optimized, for the set of resources allocated in the previous stage, according to a given robust application scheduling strategy.

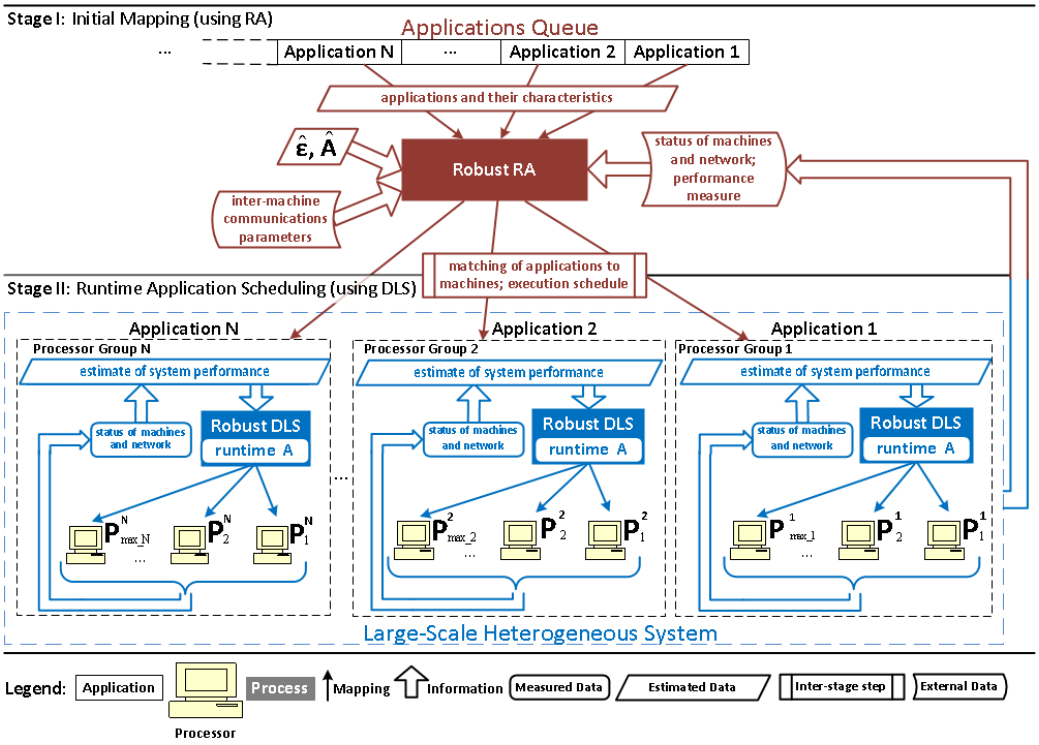

Figure 2.2: Schematic illustration of the proposed combined dual-stage framework

To claim robustness for the overall system, the following questions have been answered [5]: (i) What behavior of the system makes it robust? *Answer*: The system considered in this work is robust if all applications complete before a common deadline, given uncertainty in input data (which impacts application execution time) and system availability. The system robustness is achieved via employing robust RA and robust DLS in two consecutive stages. A robust RA is one that is capable of maximizing the probability that all applications complete before the deadline. A DLS technique is said to be robust if it facilitated the execution of an application in the smallest amount of time, *and* if this time satisfies the deadline when the runtime system availability may vary from the one assumed initially. (ii) What uncertainties is the system robust against? *Answer*: Given uncertain variations in input data and system availability, application execution times are a known source of uncertainty in the system, and may have a significant impact on the stated performance objective. The uncertainty against which the system considered in this work is assumed to be robust is the 2-tuple (π_1, π_2). (iii) How is the system robustness quantified? *Answer*: The robustness of the system, using the combined dual stage framework, can be quantified as the joint robustness of the initial mapping in stage I and the runtime application scheduling in stage II. Let ρ_1 be the largest robustness value of stage I. Also, let ρ_2 be the largest robustness value of stage II. The system robustness is quantified as the 2-tuple (ρ_1, ρ_2).

The usefulness of the combined dual stage framework has been explained via a small scale example of a batch of 3 applications and 8 processors divided into two machines. Further details of the results and the benefits of this approach can be found in [32].

Although work had been done towards formulating a mathematical description of the robustness metrics for a number of resource allocation techniques [106] and dynamic scheduling techniques [15], developing formal analytical models for evaluating the robustness of such techniques on heterogeneous parallel and distributed systems is still an open problem. Moreover, the existing analysis of the robustness of a resource allocation and a task scheduling system has been performed via simulation. Analyzing the robustness of a resource allocation via high-level models of application execution on parallel and distributed computing systems allow the computational resources to be utilized more efficiently for yielding an optimized execution performance. Therefore, the analysis process should have low computational cost (overhead). In addition, the high-level models need to represent realistic configurations of the overall system and not a simplified version. Simulation-based analysis is very time consuming and it imposes the additional burden computing confidence intervals for the results. In addition, simulation methods consider strong simplifying assumptions and approximations which compromise their accuracy. Therefore, there is a need for employing numerical analytical methods and formalism for performance modeling and analysis of the robustness of resource allocations for applications in a parallel and distributed computing environment.

2.2 Performance Modeling and Evaluation of Parallel and Distributed Computing Systems

Performance modeling is concerned with the dynamic behavior of systems and a quantified assessment of that behavior. A model can be constructed to represent some aspect of the dynamic behavior of a system. Once the model is created, it can be used as a tool

for investigating the behavior of the system. Parallel and distributed computing systems behave as discrete event systems. The state of such systems is characterized by variables which take distinct values and which change by discrete events. Therefore, at a given distinct time, the occurrence of an event within the system results in a change in one or more of the state variables. Within discrete event systems, there is a distinction between a discrete time representation and a continuous time representation. In models with a discrete time representation, the system is only considered at predetermined time intervals, such as a time sample of a number of seconds. In models with a continuous time representation, the system is considered at the time of occurrence of each event. Therefore, in such models, the time parameter is conceptually continuous. Under the realistic assumption, about events that are countably finite and that can affect the performance of parallel and distributed computing systems, are stochastic in nature and do not necessarily occur at predefined time intervals, models with continuous time representations are more suitable for evaluating such systems.

Markovian models have been shown to be an effective technique for performance analysis of computer and communication systems, where the system components are modeled as Markov processes and the overall performance (for example, throughput, resource utilization, and others) is evaluated from the numerical analysis of these Markov processes [114]. A Markov process, $X(t)$, is a stochastic process that holds a Markov property such that, $\forall\{X(t) : 0 \leq t < \infty\}$, the joint probability, $Pr(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n, \cdots, X(t_1) = x_1) = Pr(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n)$. This signifies that the future path $X(s)$ for $s > t$, does not depend upon the knowledge of the past history $X(u)$ for

$u < t, \forall (0 \leq t < \infty)$. Continuous time Markov chain (CTMC) is a mathematical model composed of continuous time stochastic processes that hold the Markov property. CTMC has been used for numerical analysis of performance of a number of computer and communication systems, such as queueing networks [81][92]. The *state-transition* diagram of a Markov process captures all the information about the states of the system and the transitions that occur between these states. Often, for the convenience of the reasoning of these processes, this state-transition diagram is represented as an *infinitesimal generator matrix*, Q. A state space size N yields an NxN matrix. A matrix element, $q(i, j)$, represents the rate at which a transition occurs between states x_i and x_j and its value is a random number drawn from an exponential distribution, which provides a memoryless property to the processes in the model. Another important feature in the derivation of performance measures is the *steady state probability distribution*. It is the probability distribution of a random variable $X(t)$ over a state space S, as the system settles into a regular pattern of behavior or *equilibrium*. For a state space size N, the steady state probability vector, π, contains N elements and each element of the vector is an unknown value that represents the steady state probability of $X(t)$ to be in one of the N states at equilibrium. In CTMC, these steady state probability values are calculated via solving the following *global balance equation* [114].

$$\pi \cdot Q = 0, \qquad s.t. \sum_{i} \pi[i] = 1 \tag{2.1}$$

Further, *reward structures* are used to derive performance measures, such as utilization and throughput, from the steady state probability values obtained from Equation 2.1 [91]. The reward associated with a component, and the corresponding state, is the sum of the rewards

attached to the activities it enables. Performance measures are then derived from the total reward based on the steady state probability distribution of that component [59]. If ρ_i is the reward associated with the component C_i, and $\Pi(C_i)$ is the steady state probability distribution, then the total reward for that component can be calculated as:

$$R = \sum_i \rho_i \Pi(C_i) \tag{2.2}$$

Many performance measures of interest correspond to some identifiable aspect of system behavior. Since the behavior of the system is associated with activities of the system components, these performance measures can be formulated by associating a reward with an activity or set of activities enabled by that component [59].

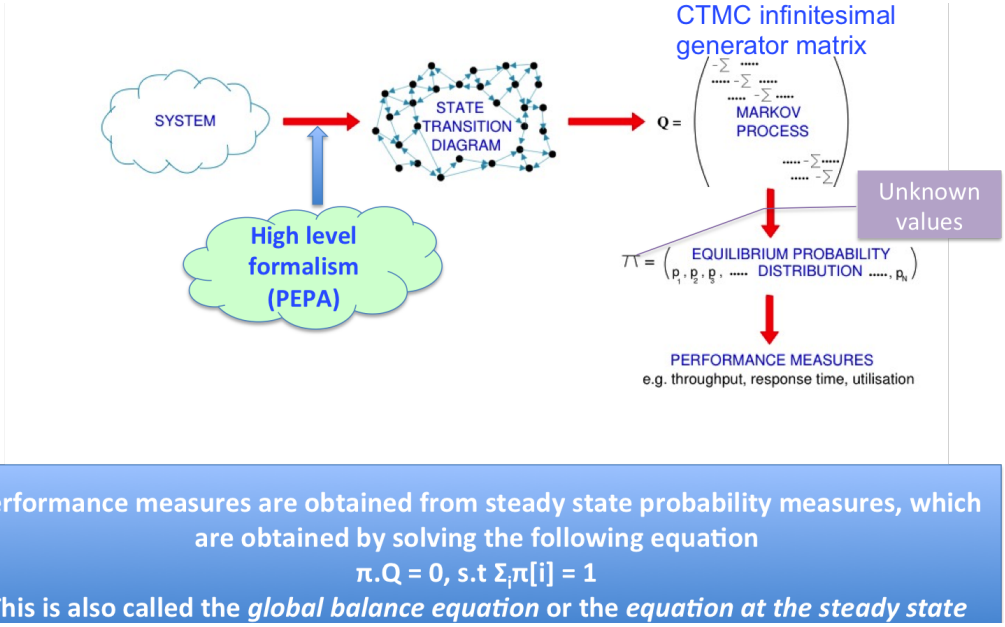

Figure 2.3: Schematic description of performance modeling and evaluation of computing systems.

Often, working directly with CTMC, at the level of state-transition diagrams and infinitesimal generator matrix, is time consuming, error prone, and infeasible for computer and communication systems that involve a large number of system components. Therefore, various high level modeling formalisms, such as queuing networks, stochastic Petri nets, and process algebras, have been developed to reduce the complexity of constructing performance models for system designers. A schematic description of performance modeling and evaluation of computing systems via a high level formalism is given in Figure 2.3. The high level formalism is further translated into the underlying mathematical structure (such as CTMCs) for obtaining performance measures.

Among the high level formalism methods, queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; and neither offer abstraction mechanisms. Process algebras, and their stochastic extensions, offer formality, compositionality, and abstraction, which are the three important features expected from a performance modeling paradigm [11][23]. Moreover, stochastic process algebras capture the timing information associated with the system components and their activities, which provides a more coherent translation of the high level formalism into the underlying mathematical structure of the CTMC. Given these advantages of process algebras over the other high level formalism methods, the work done in this thesis is focused upon the use of a suitable process algebra for modeling and evaluating the robustness of scheduling in a resource allocation system for parallel and distributed computing. A more detailed description of the theoretical developments of process algebra is given in the following sections.

2.3 Process Algebra

In theoretical computer science, process algebra (PA) is an algebraic approach to the study of concurrent processes that is used to model computer systems for obtaining qualitative and quantitative information from the modeled system. PA uses algebraic languages for the specification of processes and the formulation of statements about their role, and a calculi for the verification of these statements [10]. The term *process algebra* was coined in 1982 by Bergstra and Klop [23]. A PA is a structure that satisfies a particular set of axioms, known as the laws of PA. Given a set of atomic actions, the basic operators of the PA can be used to compose these actions into more complicated processes. Among the set of pre-defined basic operators, parallel composition is the backbone of any PA, which enables calculation and deducing qualitative results. PAs are mathematical structures that are often formulated in terms of automata, and are referred to as transition systems [23]. However, unlike automata theory, the notion of equivalence is different from language equivalence and is often referred to as the notion of bisimulation. Thus, overall PAs can be defined as the study of concurrent processes, their equational theories, transition systems, and the equivalencies between the systems.

2.3.1 Process Algebra for Parallel and Distributed Computing

Process algebra is a widely accepted and much used technique in the specification and verification of parallel and distributed software systems [23][11]. A system can be specified via the syntax provided, and the axioms can be used to verify that a computing system shows the required expected external behavior. The classical algebraic approaches to con-

currency are, Milner's Calculus of Communicating Systems (CCS) [93] and Hoare's Communicating Sequential Processes (CSP) [61]. These classical PAs provided the foundation for studying the behavioral properties of parallel and distributed computing and communication systems and enabling qualitative analysis of these systems. However, the classical PAs lacked the notion of timing and synchronization and hence could not be used for extracting quantitative information about the system due to the missing translation of the PA model to an underlying mathematical Markov model. Therefore, stochastic extensions of these classical PAs were developed that captured the timing information of the system into the PA model. Below is a detailed description of the two classical PAs, CCS and CSP, followed by a description of the stochastic PA called PEPA that evolved from the classical PAs and has been used for performance modeling and evaluation of a number of concurrent computation and communication systems. In this thesis, PEPA has been used to model and analyze performance in terms of robustness of a resource allocation in a stochastic parallel and distributed computing environment.

CCS, the Calculus of Communicating Systems is a process algebra developed by Robin Milner in 1980 [93]. The active components or processes are called agents that are built from a set of discrete actions. An action can be *observable* such that it includes an interaction or communication among different agents. Observable actions are represented by lower case alphabet letters (a, b, c, ...). Or an action may be unobservable (silent) is internal and confined to an agent. Internal actions are denoted by τ and agents can perform these actions simultaneously. Observable actions are of two types, input actions and output actions. An input action a is complimentary to an output action a'. There are primarily five

different process algebraic operators that can be used for constructing agents. Following is a description of these process constructors.

- *Action prefixing*: this is the most basic process constructor in CCS. If a is an action and P is a process, then $a \cdot P$ is a process such that P can become active only after the action a has been performed. The "$\cdot$" is an operator called *action prefixing* and denotes sequentialization.
- *Choice (+)*: If P and Q are processes, then so is $P + Q$. The process $P + Q$ has the initial capabilities of both P and Q. However, choosing to perform initially an action from P will preempt the further executions of actions from Q, and vice versa.
- *Parallel composition* ($|$): Given two CCS processes P and Q, the process $P|Q$ describes a system in which P and Q may proceed independently or may communicate via complementary ports.
- *Restriction* ($\backslash$): Let Q be a process and Σ be a set of visible actions, s.t. $\tau \notin \Sigma$. Then $(Q)\backslash\Sigma$ is a process that can execute all the actions of Q except the actions in the set Σ or their complementary actions.
- *Relabeling*: Process P is similar to process Q, where the actions of P are obtained by mapping the actions of Q through a transformation function m. Thus, P is known to be relabeled as Q.

CCS has been given operational semantics that specifies the behavior of CCS by defining a simple abstract machine for it, using a labelled transition system, similar to the style of Plotkin [94]. Further, a derivative tree or a graph can be constructed, where the language terms are represented by the nodes, and the transitions are represented by the arcs. The labelled structure is a useful tool for reasoning about agents and the systems they represent, and forms the basis for the bisimulation style of equivalence. Two agents are considered to be equivalent if they perform the exact same actions. Equivalence can be strong or weak, depending on whether the internal actions of an agent are also considered to be observable. CCS models have been used extensively to establish the correct behavior of systems, in an abstract sense, with respect to a given specification. This is also know as functional

or qualitative modeling where, behavioral properties such as fairness and freedom from deadlock are investigated, in contrast to quantitative modeling where, the quantitative values extracted from performance models. IN CCS, no timing information is associated with processes and their actions. Therefore, quantitative values cannot be extracted from system models derived using CCS.

CSP, the Communicating Sequential Processes, a classical process algebra that evolved from CCS was introduced by Hoare, 1984 [61]. It is an abstract and formal event-based language to model concurrent systems. CSP was developed for a specific group of researchers who wanted a simpler modeling language with a smaller set of operations than CCS and the main objective behind developing CSP had been to find the simplest possible mathematical theory with the following desirable properties [61]:

- It should describe a wide range of interesting computer applications, from vending machines, through process control and discrete event simulation, to shared-resource operating systems.
- It should be capable of efficient implementation on a variety of conventional and novel computer architectures, from time-sharing computers through microprocessors to networks of communicating microprocessors.
- It should provide clear assistance to the programmer in his tasks of specification, design, implementation, verification and validation of complex computer systems.

Simplicity was the motivation behind developing CSP and was sought through designing of a single simple model, such that it is easy to define as many operators as possible for appropriate investigation of a range of distinct concepts. In CSP, a concurrent system is made of a set of interacting *processes*. Each process sequentially produces *events*. Each event is atomic and the set of all events belonging to a process is called an *alphabet*. The basic constructs or the operators of CSP are described below.

- *Prefix*: $a \rightarrow P$ is the prefix construct for a process P and an event a belonging to the alphabet of P, such that a process upon performing a behaves as P.
- *Choice*: Unlike CCS, where the Choice operator always presented a deterministic choice, the construct in CSP is able to provide the functionality of a system with both a deterministic and a non-deterministic choice.
  - *Non-deterministic choice* (Π): If a system is defines as, $R_1 = a \rightarrow P \Pi b \rightarrow Q$, the choice between P and Q is decided internally by the system itself. The environment, external to the system, has no control over the choice.
  - *Deterministic choice*($+$): Here if a system is described as, $R_1 = a \rightarrow P + b \rightarrow Q$, then the system can behave as either P or Q. The choice is decided by the external environment depending on the actions offered by the environment.
- *Parallel Composition* ($\|$): $P \parallel Q$ denotes a process that behaves as the interleaving of P and Q, but synchronizing them on the events that are common to both P and Q.
- *Hiding (abstraction)* ($/$): Denoted by P/A, where A is the alphabet of the events of P that are not visible to the external environment.

CSP has been used in a modeling a number of modern computing systems such as, software design of dependable and safety-critical systems, fault management system to confirm that the design is free of deadlock, systems that incorporate complex message exchanges, verification of communications and security protocols, and verification of elements in communication systems to verify their correctness. The minimized operator set (in comparison to CCS) is useful in theoretical investigations of modern computing systems [61]. However, because a process must specify the names of all the other linked processes, it is hard to write large libraries of functions for very large computing systems with a small set of operators. Moreover, like CCS, there is no timing information associated with processes and their events in CSP. Therefore, quantitative values cannot be extracted from performance models designed using CSP.

2.3.2 Performance Evaluation Process Algebra

The PEPA project was initiated at the Laboratory for Foundations of Computer Science, a research institute of the Division of Informatics at the University of Edinburgh, by Jane Hillston in 1991 [59]. The project was motivated by the problems that were encountered during the performance analysis of large computer and communication systems via the numerical analysis of the underlying Markov processes and the CTMCs. PEPA provided a high level formalism for CTMC, which expresses the behavior or performance of the computing system. The performance measure is obtained from the CTMC generator matrix and the steady state probabilities. PEPA is an expressive formal language for modeling distributed systems. PEPA models are constructed by the composition of components which perform individual activities or cooperate on shared ones. To each activity is attached an estimate of the rate at which it may be performed. Using such a model, a system designer can determine whether a candidate design meets both the behavioral and the temporal requirements demanded of it. PEPA offers several attractive features which were not available in performance modeling paradigms, such as queueing networks and Petri nets, that existed prior to process algebras. The most important of these features are:

- *compositionality*, which is the ability to model a system as the interaction of subsystems,
- *formality*, which means giving a precise meaning to all terms in the language, and
- *abstraction*, which is the ability to build up complex models from detailed components, disregarding the details when they are not necessary.

Queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; and neither offer abstraction mecha-

nisms. In addition, when compared to classical process algebras, such as CCS and CSP, PEPA provides the modeling of timing information associated with the components of the system and their activities, which enables extraction of quantitative information from the PEPA models of the computing systems.

It is shown that this language supports a compositional approach to model construction, resulting in models which are easy to understand and modify. Moreover, the structure provided within a model can be exploited for model manipulation and simplification [59][60]. The objective behind developing a language in which the performance evaluation features can be regarded as an extension to the classical process algebra to be used as a design formalism within the performance model. The features of the classical process algebra that are regarded as being essential are described below.

- *Parsimony*: Process algebras are simple languages with only a few elements. Therefore, it provides easy reasoning and a great deal of flexibility to the modeler. In PEPA the basic elements of the language are components (similar to agents in CCS and CSP) and activities (similar to actions in CCS and events in CSP) that correspond to states and transitions in the underlying stochastic model.
- *Formal definition*: The language is given a structured operational semantics in the style of Plotkin [94], providing a formal interpretation of all expressions. The notions of equivalence which are subsequently developed are based on these semantic rules. This gives a formal basis for the comparison and manipulation of models and components, and introduces the possibility of developing tools to automate, or semi-automate, these tasks.
- *Compositionality*: The model structure provided by the compositional nature of process algebras, and the ability to reason about that structure, have been highlighted as a major motivation for investigating the use of such a language for performance modeling. In PEPA the cooperation combinator forms the basis of composition. Model simplification and aggregation techniques can be developed which are complementary to this combinator. This means that part of a model can be simplified in isolation, if its interaction with the rest of the system is modeled by such a combinator, and replaced by the simplified component without affecting the integrity of the overall model.

The main attribute which is missing from a classical process algebra such as CCS or CSP, and which is necessary for performance evaluation, is the quantification of time and uncertainty. In performance models, in order that performance measures can be extracted from the model, it is important that timing behavior and uncertainty be quantifiable. This is achieved in PEPA by associating a random variable with each activity of a component, representing its duration. A delay is thus incorporated in each activity in the model and the timing behavior of the system is captured. Moreover since the duration is a random variable, temporal uncertainty, which is the uncertainty of how long an action will take, is represented. This also captures spatial uncertainty, the uncertainty about what will happen next within a system, which is an inherent property of parallel and distributed systems. Thus adapting the process algebra to make it suitable for performance modeling is achieved by introducing a random variable for each activity within the system. The construction is analogous to the association of a duration with the firing of a timed transition the stochastic extensions of Petri nets.

The theoretical developments underpinning PEPA has been focused on the interaction between process algebra and the underlying mathematical structure, the Markov process. This work has been broadly classified into three areas, (i) designing the language, (ii) manipulating the models, and (iii) solving the models and deriving performance measures. For designing the language, PEPA models are described as interactions of components. Each component can perform a set of actions: an action $a \in Act$ is described by a pair (α, r), where $\alpha \in A$ is the type of the action and $r \in R+$ is the parameter of a negative exponential distribution governing its duration. Whenever a process P can perform an action,

an instance of the probability distribution is sampled: the resulting number specifies how long it will take to complete the action for that process. The components correspond to the identifiable parts in the system, or roles in the behavior of the system. They represent the active units within a system; the activities capture the actions of those units. For example, a job queue may be considered to consist of an arrival component, which defines the jibs arriving at the queue with a job arrival rate, and a service component that describes the jobs being serviced from the queue at a given service rate. The components interact to form the behavior of the job-queue system. A component may be atomic or may itself be composed of components. Thus, the queue in the above example may be considered to be a component, composed of the atomic components, arrival and service. Assuming that there is a countable set of possible components, C, each component has a behavior which is defined by the activities in which it can engage. From the example given above, actions of the queue might be *accept*, when a job enters the queue, *service*, when a job is being serviced from the queue, or *loss*, when a job is denied entry into a full queue. Like CCS or CSP, to model situations when a system is carrying out some action (or sequence of actions) which is unknown or unimportant, there is a distinguished action type, which can be regarded as the unknown type, in PEPA and is represented by the symbol τ. Activities of this type will be private to their associated component.

A small but powerful set of combinators is used to build up complex behavior from simpler behavior. The set of combinators in PEPA provide an extension to the classical process algebra. These combinators are explained in detail below.

- *Prefix* ($\cdot$): this is defined as the designated first action and represents a sequence of activities performed by a component. For a component P with an activity α and an activity rate r, prefix can be used to define the component as $P ::= (\alpha, r) \cdot P$.
- *Choice* ($+$): this combinator provides a choice between two competing components, which is often determined by a race policy dependent on the apparent rates of the activities of those components.
- *Cooperation* ($\underset{\mathcal{L}}{\bowtie}$): this combinator defines a set of components that can perform their activities concurrently and synchronize on the activities that belong to the co-operation set $\mathcal{L}$. For example, components P and Q, can be defined as $P \underset{\mathcal{L}}{\bowtie} Q$, where α is a *concurrent activity* if $\alpha \notin \mathcal{L}$, and is a *cooperative activity* if $\alpha \in \mathcal{L}$.
- *Hiding* ($/$): This combinator provides an abstraction of the range of activities that can be considered for a component during the analysis of that component. In this case, a component P can be defined as $P ::= P/L$, where L is the set of activities that are hidden from or unknown to the component P. Therefore if an activity $\alpha \in L$, then for P, $\alpha = \tau$.

The semantic rules of PEPA generate a labelled transition system, similar to classical process algebra. However there are some significant differences introduced by the inclusion of quantified information in PEPA. The semantics may give rise to a multi-transition system. Moreover, it is not sufficient to record the existence of a transition or arc between two nodes, as the rate at which the transition occurs also becomes significant part of the analysis. The multiplicity of the transition is important. This is because the apparent rate of a term which has two copies of the same arc, generated by two instances of the same action, will differ from that of a term with only one instance. The generated derivation graph forms the basis of the Markov process on which performance analysis will be carried out. Driven by the problem of state space explosion, there have been efforts to address the problem using PEPA for aggregating Markov processes by considering a coarser view of the state space, grouping equivalent states into clusters, and redefining the dynamics of the system [59][60].

To manipulate models (via model simplification and model aggregation) there is a need to define equivalence relations. In process algebra, equivalence relations are based on the notion of observability. In PEPA, observation is assumed to record timing information over a number of runs. The equivalence in PEPA models is defined via *bisimulation* as opposed to *trace equivalence* in other formal languages. Two trace equivalent components, p and Q, of a PEPA model may not always be bi-simulation equivalent, because at a given time, the number of states associated with P and Q may not always be equal. PEPA will identify this difference between the states of the components and will not aggregate them under the same label. The equivalence relations are used to manipulate PEPA models via (i)*model simplification*, which uses model-model equivalence if different models imply the same pattern of partial correlations among the variables in them, to substitute complex model with a simpler model from a solution point of view. (ii) *Model aggregation* that uses a state-state equivalence to establish a partition of the state space of the model and replace a large number of equivalent states with one *macro state*.

After the models have been manipulated to simplify the state space from a solution point of view, there is a need for solving the model to derive performance measures from the underlying CTMC structure. A state of the Markov process is associated with each node of the state-transition graph generated from the PEPA model, and the transition rate between two states is the sum of the rates of the actions labeling the arcs between the corresponding nodes. PEPA models contain information about the duration of activities and their relative probabilities. From these models a corresponding continuous time Markov chain (CTMC) is generated by elaborating the model against the structured operational se-

mantics of the PEPA language. The CTMC processes can be translated into an infinitesimal generator matrix (explained in earlier section of this chapter), where the rows and columns of the matrix represent the states of the processes or PEPA components, and each element of the matrix represents the rate of the transition between the two associated states of the model. This rate of transition is the same as the rate of the activity that leads a component to transform from one state to another in the corresponding PEPA model. Linear algebra can then be used to solve the model in terms of equilibrium behavior. This behavior is represented as a probability distribution over all the possible states of the model, which is also represented as the steady state probability distribution vector. This vector consists of unknown steady state probability values that can be obtained by solving the global balance equation. However, this distribution is seldom the ultimate goal of performance analysis. The modeler is interested in performance measures which can be derived from this distribution via a *reward structure* defined over the CTMC [59].

A range of tools have been developed to solve PEPA models for performance analysis. The premier PEPA tool is the PEPA Workbench Eclipse Plug-in [53], which was also the first tool developed as a part of the PEPA project for performance analysis of distributed computer and communication systems and for deriving performance measures such as, utilization and throughput. More recently support for the PEPA language has been added to other tools such as the Mobius Modeling Framework, developed by the Performability Engineering Research Group, Motorola Center for High-Availability System Validation at the University of Illinois at Urbana-Champaign. PEPA is also supported by the PRISM probabilistic model checker at the University of Birmingham, England. The latest devel-

opment in the suite of PEPA tools is the Imperial PEPA Compiler (IPC) [24]. It is a tool that extends the modeling capability, provided by PEPA, by allowing extraction of a larger number of performability measures (such as, performance and reliability metrics) from the PEPA models. It compiles PEPA models to Will Knottenbelt's DNAmaca format [24], which is well suited for storage and analysis of very large systems. In particular, DNAmaca is adept at analyzing very large Markov models, and producing passage-time distributions and reliability quantiles or bounds. The analysis of Markov models for generating passage time distributions, to capture the time elapsed between the start and end of any two activities, has also been added as a functionality to the premier PEPA Workbench [53].

PEPA and the aforementioned PEPA tools have been used for performance modeling and analysis of a wide range of concurrent systems. In a recent research study related to the scheduling of pipeline applications on grid resources, the PEPA workbench [53] has been to solve the performance models of such a scheduling system to obtain relevant performance information required for enhancing the execution performance of pipeline applications on the allocated grid resources (processors and network links) [19][21]. In this work, the authors use algorithmic skeletons that describe the intended execution path of the pipeline application on a grid computing system. PEPA language is used to model the pipeline structured skeletons, where the problem is split into modeling the stages of the pipeline, the processing resources, and the network resources as components within the model. The overall model is defined by modeling the mapping of the application on grid resources using the cooperation combinator between the pipeline stages, the processors, and the network. The PEPA workbench is used to calculate the performance feature, which

is the throughput of the pipeline application, that is obtained when employing the modeled mapping for scheduling the pipeline applications onto the grid resources. This measure of the throughput value is then used for comparing the performance of various possible mappings for selecting the mapping that gives the highest throughput value [19][21]. Although CTMCs provide accurate and efficient solution procedures, the CTMC models do not scale well to represent Grid-scale computing with large numbers of jobs executing on large compute clusters. In a recent research study, an approach to the problem of scalability has been provided as a continuous approximation of the discrete state space underlying the mathematical representation of the model rather than singly representing each of the individual states of the various model components. The authors propose to model a Grid-based system with the PEPA process algebra, and generate a set of ordinary differential equations (ODEs) that are used to generate the underlying mathematical structure to describe the evolution of each model component as a function of time. Solving a system of ODEs has low computational cost and memory requirements [20]. In addition, unlike CTMCs, modeling with ODEs does not require the assumption for the system to reach a steady-state equilibrium for deriving performance measures from the mathematical structure [20]. The required transformation tools that translate a PEPA model into a mathematical system of ODEs, and the numerical computing platforms that offer high-level support for the solution of the generated ODEs are provided by the IPC tool [24].

Performance measures, such as throughput and resource utilization, have successfully been analyzed, via performance modeling of application (pipeline) scheduling and grid computing systems using the PEPA process algebra, in the literature discussed above

[59][60][19][21][20]. However, to the best of our knowledge, response time measure, which is a very important measure for analyzing the performance of a resource allocation or a task scheduling system in parallel and distributed computing that are bound by time constraints (such as an execution deadline), have not been evaluated in any of these research work. The CTMC steady state analysis of the mathematical structure underlying the PEPA models (using tools such as, PEPA workbench and IPC) enables the calculation of performance measures, such as throughput and response time, using the theory of reward structures. However, it does not provide any platform for the calculation of response time measures [59]. A research direction towards evaluating response time profiles has been given in the work done on evaluating PEPA models via ODE analysis [20]. Further, an extension of this work was presented as a research study for evaluating response time profiles from *passage-end analysis* for service-based systems (for example, emergency response service quality systems for roadside assistance) [33]. Using a passage-end analysis, the authors proposed the calculation of the probability of completing a passage by a cached response at or within a given time, along with the cumulative distribution function (cdf) and probability mass function (pmf) of all the requests that were serviced successfully. In addition, the authors propose the use of *stochastic probes* that are added to a model (in the context of PEPA, it is a single sequential component) for observing and reasoning about the activities of the model. The functionality for the passage-end analysis has been implemented in the IPC tool [24] and more recently in the PEPA workbench [53]. Further, the performance specification and evaluation of response time measures, in a complex distributed wireless network system, via stochastic probes using *grouped* PEPA (GPEPA),

which is an extension of PEPA that consists of a number of labeled *cooperating* component groups composed of a large number of components executing in parallel, and *immediate* GPEPA (iGPEPA), which adds immediate action functionality to the GPEPA formalism [57], has been presented in [58]. The *immediate actions*, in an iGPEPA formalism, are high priority actions and are performed instantaneously before any other timed actions. In addition, the authors demonstrate the analysis of the response time measures using a *fluid analysis* approach, which is based around fast solutions of a system of differential equations.

In the rest of the thesis, a performance model, derived using the PEPA, GPEPA, and iGPEPA formalism for the evaluation of the robustness of resource allocations of applications in a parallel and distributed computing system, has been presented. A detailed description of the work that is related to and evolved as the modeling study presented in this thesis is given in the next chapter. The analytical model of the resource allocation system, the analysis of the response time measure (as a makespan of the system), and an analysis of the robustness the resource allocation mapping, is described in Chapter 3. An analysis of the experimental results obtained using the PEPA workbench is presented in Chapter 5. The benefits and the usefulness of the proposed model, the conclusions derived from the proposed model and the related performance analysis, along with its implications towards potential future work, is discussed in Chapter 6.

CHAPTER 3

RELATED WORK

The modeling study presented in this dissertation has resulted from a series of related work as preliminary research contributing towards the evolution of the research presented herein. This work evolved from a long study of dynamic load balancing (DLB) methodologies, such as dynamic loop scheduling (DLS), a study of robustness of scheduling and resource allocation methods used in parallel and distributed computing systems, a study of autonomic computing systems and the use of machine learning techniques for autonomic selection of robust scheduling methodologies to execute scientific applications on parallel computing environment that is prone to unpredictable variations in application and systemic characteristics. A synopsis the preliminary work that has led to the research presented in this dissertation is given in following sections.

3.1 Enhancing the Functionality of a GridSim-based Scheduler for Effective Use with Large-Scale Scientific Applications

The work presented in this section is related to the understanding of the need for DLB for executing scientific applications in parallel and distributed computing systems [108]. A number of DLS techniques were surveyed to understand the utlization and the benefits of using DLS for executing scientific applications on a parallel and distributed computing sys-

tems, which are prone to load imbalance at runtime. A number of DLS techniques that have been developed to provide load balancing on parallel and distributed systems for the execution of scientific applications with large, computationally intensive loops and irregular loop iteration execution times, were surveyed. Further details regarding the experimental scalability studies of the DLS techniques such as, fixed sized chunking (FSC), guided self scheduling (GSS), factoring (FAC), weighted factoring (WF), adaptive weighted factoring (AWF) and its variants AWF-batched (AWF-B) and AWF-chunked (AWF-C), adaptive factoring (AF) can be found in the literature [78] [95] [66] [64] [27] [13] [15].

3.1.1 Motivation

In previous studies, these DLS methods have been tested for a number of scientific applications running on ready to use infrastructure for parallel and distributed systems. However, the experiments conducted involved a limited range of problem sizes and number of processors available for running the scientific applications. This also limited the testing of the DLS methods for their scalability and adaptability. The scalability of the DLS techniques has been previously studied experimentally on various architectural and computational environments with limited number of processors. Analyzing the scalability of these DLS methods and scheduling scientific applications on large scale parallel systems requires a framework for modeling and simulation of such systems and the scientific applications. To address and overcome the limitations of testing the DLS methods exhaustively on real systems, we use an event based simulator called Alea [76]. It is a task scheduling simulator built on top of the GridSim simulator [25]. Alea is composed of independent

entities which communicate amongst each other through message passing. To the best of our knowledge, the DLS methods are implemented for the first time in a simulator in this work [108]. In contrast to the scheduling techniques already present in Alea, DLS address the variations in the algorithmic and systemic characteristics during application execution. The first step for running the experiments is to generate the application tasks (jobs) and their characteristics. We used a workload generator called Lublin [85] for generating a workload file in the standard workload format (SWF) [30]. The workload file carries the `.swf` extension and consists of the specified number of jobs or tasks of the application and their characteristics. The execution times of these tasks follows a hyper-gamma distribution to simulate the irregular loop iteration execution times within a scientific application. The hyper-gamma distribution has already been implemented in Lublin [85] to generate the running times of various tasks. This workload file is then read by the job submission system to create job descriptions in the form of gridlets and to send these gridlets to the scheduler entity.

3.1.2 Integrating DLS within Alea

The first step for running the experiments is to generate the application tasks (jobs) and their characteristics. We used a workload generator called Lublin [85] for generating a workload file in the standard workload format (SWF) [30]. The workload file carries the `.swf` extension and consists of the specified number of jobs or tasks of the application and their characteristics. The execution times of these tasks follows a hyper-gamma distribution to simulate the irregular loop iteration execution times within a scientific application.

The hyper-gamma distribution has already been implemented in Lublin [85] to generate the running times of various tasks. This workload file is then read by the job submission system to create job descriptions in the form of gridlets and to send these gridlets to the scheduler entity. The grid resource entity reads the number and characteristics of the resources from a machine file which is generated offline along with the workload file. The machine file carries the extension `.swf.machines` and contains a list of all the grid resources and their respective configurations. Alea allows a resource to be configured with several machines, and each machine to be configured with several processors. As Alea only allows mapping of one task to one processor at a time, it was a challenging task to implement the DLS methods by which a chunk of tasks is assigned to a processor at a time. To overcome this problem, we configured each grid resource to consist of a single processor by declaring in the machine file only one machine and a single processor to a grid resource. This enabled assigning more than one task to a resource or, in our case, a processor. The implementation of the DLS methods in Alea requires their incorporation into the scheduler module, which is the `Scheduler.java` class. After the scheduler assigns and executes the application tasks onto the resources according to the scheduling policies provided by the DLS methods, it sends the execution results, such as the makespan value, the resource utilization, the scheduling overhead, and others, to the results collector entity, which in turn, reports these statistics to the user.

3.1.3 Analysis of simulation results

In this work, an analysis of the makespan value and the resource utilization has been illustrated Figures 3.1 and 3.2 to evaluate the performance and scalability of the DLS methods. These results are in confirmation with the analytical and experimental results obtained for these DLS methods in previous studies [78] [95] [66] [64] [27] [13] [15].

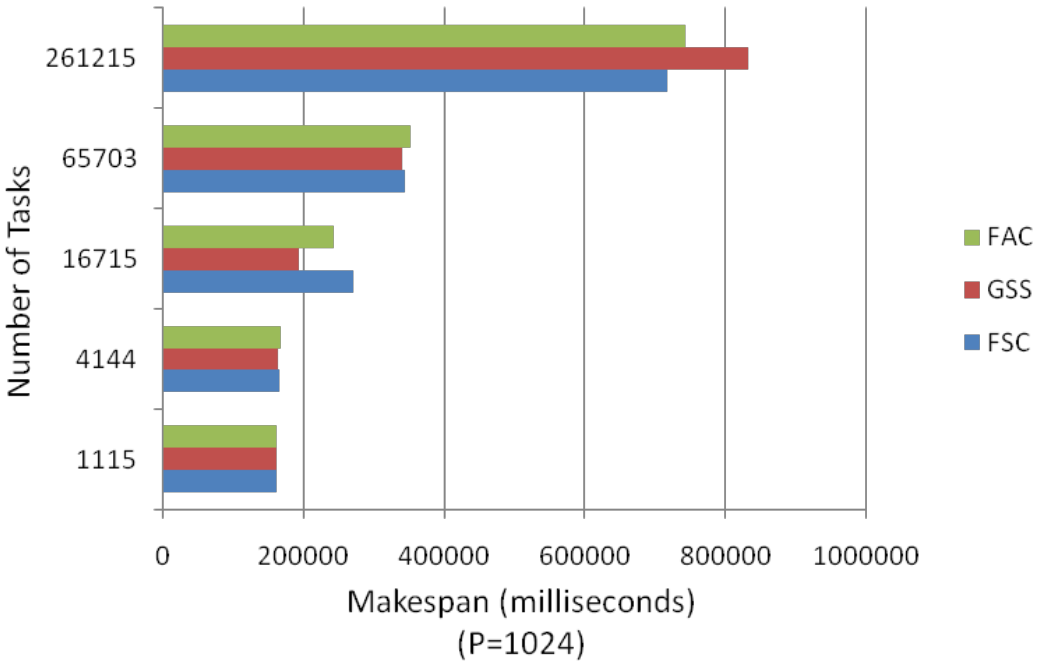

Figure 3.1: Makespan obtained from simulated execution of different number of tasks on 1024 resources.

3.2 Performance Optimization of Scientific Applications using an Autonomic Computing Approach

An effective approach for improving the performance of scientific applications via autonomic computing using machine learning techniques (reinforcement learning (RL)) is

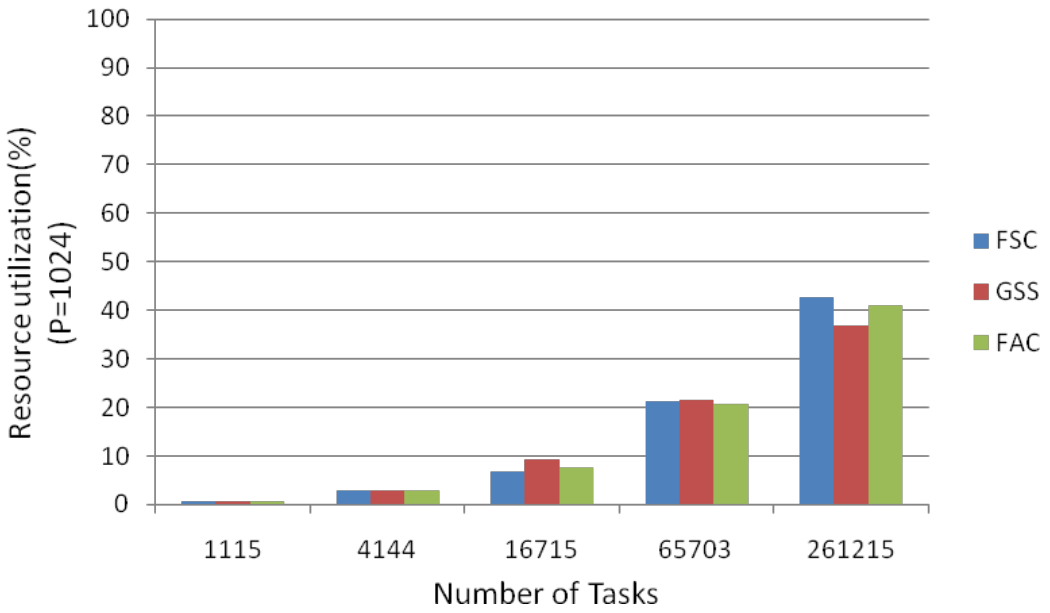

Figure 3.2: Resource utilization of simulated execution of different number of tasks on 1024 resources.

described in this section [16]. This study was conducted to understand the the use of machine learning techniques for an autonomic selection of DLS methodologies based on their performance for scheduling time-stepping scientific applications on high performance computing systems.

3.2.1 Motivation

Selecting an effective and efficient scheduling algorithm from the currently available options to achieve load balancing for applications executing in an unpredictable environment is a difficult task. The difficulty is due to the complex nature of application characteristics, which may change during runtime, combined with the dynamic nature and unpredictability of the computing environment. Load balancing may be necessary in several parts of an application, and each part may require different scheduling algorithms for

optimal performance. Furthermore, certain scientific applications require the execution of their computations repeatedly over the computational domain. The repetitive calculations are usually performed over a series of time-steps. Such applications are referred to as time-stepping applications, and examples include heat solvers, solving time-dependent Euler equations, N-Body simulations, simulation of wavepackets dynamics, and others. In applications requiring a large number of time-steps, the load imbalance characteristics of each part may vary as the application execution progresses through the time-steps. A scheduling algorithm selected *offline*, which performs well early in the application's lifetime, may later become inappropriate. Therefore, in this scenario, the selection of a scheduling algorithm for such a dynamic environment is a very difficult task, and a relatively more intelligent entity is needed to dynamically select during runtime the best scheduling algorithm for (possibly each part of) an application. In this work, the autonomic computing aspect of the execution of an application is focused on application's self-management attributes with respect to performance optimization. Therefore, the dynamic selection of the DLS algorithms must be based on the application performance during its execution. A RL agent implementing two RL techniques, Q-learning and SARSA [115], has been integrated with a scientific parallel application characterized by a large number of time-steps (a time-stepping application) [36, 37].

3.2.2 Integrated framework for an autonomic algorithm selection

In [36], a RL agent incorporating Q-learning and SARSA was embedded into a parallel scientific application, namely simulation of wavepacket dynamics using the quantum tra-

jectory method (QTM). To our best knowledge, [36] is the first work that has attempted to integrate an RL agent with a parallel scientific application for *autonomic* DLS algorithm(s) selection. Subsequent works investigated and reported on a performance comparison of the QTM application in terms of T_p *with* and *without* the RL agent, with varying learning rate (α) and discount factor (γ), and the influence of a particular RL technique for a particular set of learning parameters (α,γ) [37, 96]. The algorithm selection problem is addressed by providing a generic design of a RL system to *autonomically* determine at runtime the optimal scheduling algorithm for a time-stepping application using RL techniques. Figure 3.3 illustrates the design of the proposed RL system, derived by adding the loop scheduling context to the environment of a generic RL system. During the first few invocations of the loop, the agent simply specifies each algorithm in the library in a *round-robin* fashion, in absence of prior knowledge about the characteristics of the loop. When sufficient knowledge is obtained during this initial learning period, the agent applies an adaptive learning *policy* B (Q-learning or SARSA) on the accumulated information to select an algorithm (action $a-$select a particular DLS method) from the library of DLS algorithms, and the environment moves to another state s (application is using the particular selected DLS method). The loop completion time using the selected DLS algorithm determines a performance level, which is the basis of the reward r for the action a taken by the RL agent. The application communicates information i about the changed state s and the reward r to the RL agent, for continuous learning by the policy B. If the agent takes action only after a specified number of loop invocations, the application simply reuses the algorithm

associated with the current state s, denoted by the loopback arrow from the environment to the library (Figure 3.3).

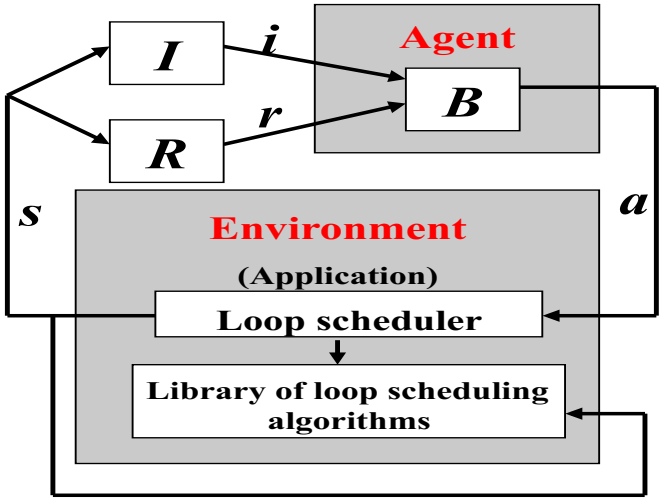

Figure 3.3: RL system for *autonomic* selection of DLS methods

3.2.3 Experimental results, analysis, and evaluation

Following an analysis of the results in Figure 3.4, the following observations can be made:

- For each p, the T_p of the application with either RL technique (from the LEARN set) is significantly lower than the T_p of the application without learning (i.e., a DLS method from the NOLEARN set).
- For each p, there is no significant difference between the T_p obtained using Q-learning or SARSA.
- For the LEARN set, there is a significant drop in the T_p when p is increased from p=2 to p=8. T_p does not significantly change, however, as p is further increased from p=12 to p=24. When using RL the optimum p for the application with 501 pseudoparticles is $p*$ = 12.
- For the NOLEARN set, STATIC has the worst T_p from p=2 to p=8, but better T_p than most other techniques in NOLEARN for p=16 to p=24. The explanation is

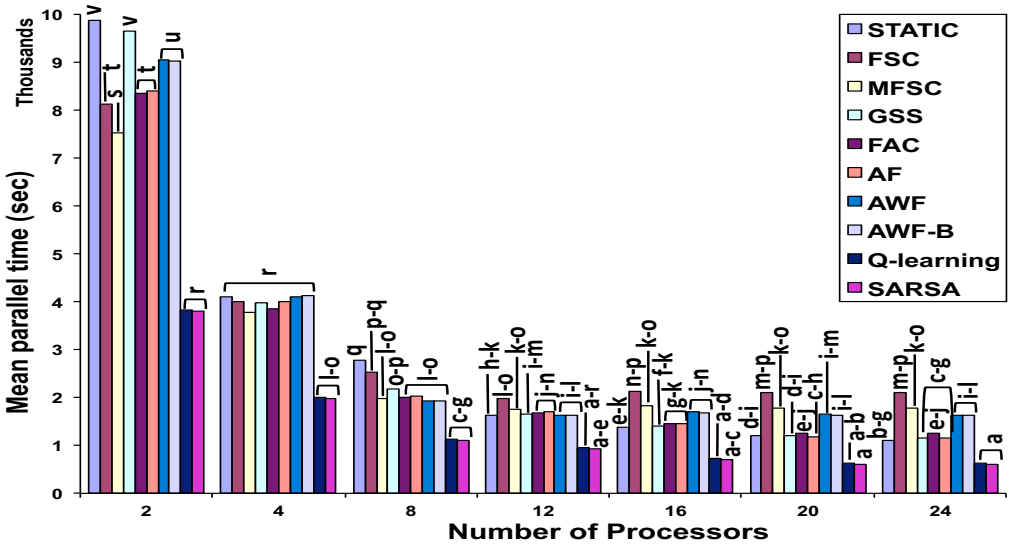

Figure 3.4: Mean parallel time (T_p) for wavepacket simulation using QTM using with RL

that with a fixed problem size, the performance of a dynamic scheduling method degrades with additional processors due to the increase in scheduling overhead. It is well known that STATIC has no scheduling overhead, therefore, it is not penalized as the dynamic techniques when using more processors.

- The T_p for the LEARN set at p=2 is not significantly different from the T_p for the NOLEARN set at p=4. Similarly, the T_p for LEARN at p=4 is statistically comparable to the T_p for NOLEARN at p=8, with the exception of STATIC and FSC. For $p \leq$8, the QTM application using RL on p processors has statistically the same T_p as the application without RL on the next higher p. The T_p for the LEARN set at p=12 is even significantly better than the T_p for the NOLEARN set at p=16.

These results validate the suitability of RL as a viable procedure for online selection of DLS algorithms from a library to improve the performance of a class of large, time-stepping scientific applications with computationally intensive parallel loops.

3.3 Investigating the robustness of dynamic loop scheduling on heterogeneous computing systems

This study was one of the first steps towards defining robustness for dynamic scheduling methods and formulating robustness metrics to guarantee certain performance level of such DLS methods, to measure their robustness against various unpredictable variations of factors in the computing environment [107].

3.3.1 Motivation

Scientific applications often contain large loops. Running such computationally intensive applications in heterogeneous environments exhibits an irregular behavior, in general due to both variations of algorithmic and systemic nature. Therefore, load imbalance is their major performance degradation factor. Heterogeneous computing systems are uncertain computing environments and often consist of computing resources that differ in quantity and availability over time. Nowadays, time to solution consists of more factors than just the execution time of the application. It naturally follows that new metrics are needed to characterize the time to solution, in addition to the traditional performance metrics, such as execution time, efficiency, scalability, and others. The new metrics must characterize the robustness of scientific applications running on the complex high-performance computing systems. Therefore, a study of robustness is essential to ensure the performance of the techniques used to parallelize scientific applications under highly unpredictable conditions. DLS techniques are one of the best ods to obtain dynamic load balancing, because DLS methods are based on probabilistic analyses and are inherently robust against perturbations at runtime. In this work, two metrics are proposed to measure the robustness of

these techniques against variations of two system related parameters: *load variations* and *processor failures*.

3.3.2 Formulating robustness metrics for DLS

The study in [107] shows that the FePIA procedure [5] can be used to derive metrics to model and estimate the robustness of the DLS methods against various perturbation parameters. This procedure consists of the four steps, (i) *Identify the performance features.*, (ii) *Identify the perturbation parameters.*, (iii) *Identify and clarify the impact of perturbation parameters on performance features*, (iv) *Identify the analysis to determine the robustness*. An increased system load imbalance and the presence of failures are expected to degrade the performance of the DLS techniques. Possible such scenarios are illustrated in Figure 3.5 and are elaborated later in the paper. We formulate here a *flexibility metric* as a measure of robustness against system load variations, and *resilience metric* as measure of robustness against processor failures.

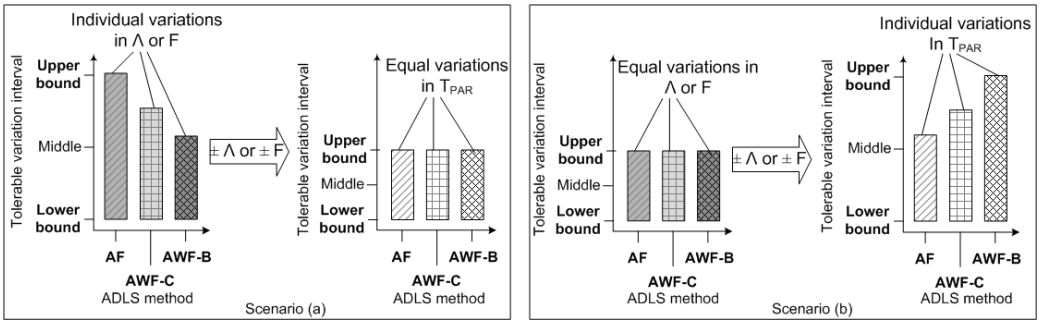

Figure 3.5: Two possible scenarios to determine DLS flexibility

Let us assume N independent tasks. Each loop iteration is considered to be a task and the terms iteration and task are used interchangeably. The goal of any DLS technique is to schedule the N tasks onto the set of P processors of large-scale heterogeneous computing systems, while minimizing the total parallel execution time (or *makespan*) T_{PAR}. A minimum T_{PAR} is achieved via dynamic load balancing, and accounting for different processor speeds. Each processor executes a set of tasks (called chunks) at a time. Each task is executed in a non-preemptive fashion, i.e., no other tasks of higher priority will suspend it. The same holds for the execution of a chunk, or for all chunks during a single time-step. In this work, the *performance features*of interest are: *the (expected) processor finishing time*, $\mathcal{ET}_j$, *the total parallel time*, T_{PAR}, and *the number of iterations that need to be rescheduled*, N_{resch}. The performance features should be limited in variation under certain application, system or environment related perturbations parameters. The *perturbation parameters* of interest include variations of the following: irregularities of application computational requirements, system availability due to uncertain load variations, and resource reliability (caused by processor or network failures). Commonly, all these perturbation parameters vary over time and cause uncertainties during runtime. A robust DLS algorithm must adapt to any kind of variations in these perturbation parameters, while constraining the variation in the performance features. Designing robustness metrics that incorporate *all* these parameters is very challenging [5].

To measure the flexibility of DLS against perturbations in system load, we formulate here the *flexibility metric* to measure the robustness of DLS methods against *system load perturbations* We make the following assumptions with respect to perturbations in the load

of the heterogeneous system during run-time: (i) variations of individual worker loads are mutually independent, (ii) individual worker loads may or may not occur simultaneously, (iii) DLS has load variation detection and monitoring mechanisms.

In Step 1, let $\Phi = \{\phi_1\}$, where $\phi_1 = \mathcal{ET}_j$ and $1 \leq j \leq P$ be the performance features set. The individual finishing time, $\mathcal{ET}_j$, of processor m_j is the sum of *computation* times T_j, of all tasks a_i executed by m_j, and the sum of *communication* times T_j^{W2F}, between processor m_j and its corresponding foreman, and the sum of *communication* times T_j^{W2W}, between m_j and any other worker processor. Mathematically, for all $\{$tasks $i|a_i$ executed on $m_j\}$, this is written as:

$$\mathcal{ET}_j = \sum_{i,j}^{N,P} \left(T_j + T_j^{W2F} + T_j^{W2W}\right) \tag{3.1}$$

In Step 2, let the perturbation parameters set be $\Pi = \{\pi_1\}$, where $\pi_1 = \lambda_j$ and $1 \leq j \leq P$. For the analysis in this work, it is considered that the perturbation parameter is to be λ_j, times the *individual* load of processor m_j. Vector λ contains the load values of all processors in the target system. DLS *initially assumes* that the system has λ^{orig} load. The value of λ^{orig} can be usually determined by executing the first batch of chunks, as determined by the original factoring rules and their subsequent evolution. The initial load of m_j is λ_j^{orig}, found at position j in the λ^{orig} vector.

In Step 3, in order to determine the impact of λ_j over $\mathcal{ET}_j$, for all processors, their finishing time, given their *own* load, is analyzed individually. Each actual finishing time is expected to vary with according to λ_j. This is denoted as $\mathcal{ET}_j(\lambda_j)$ – the actual execution (computation and any communication associated with it) time of all tasks a_i assigned to

m_j, in the presence of load variation on m_j, as indicated by λ_j. Mathematically, for all $\{$tasks $i|a_i$ executed on m_j under varying load $\lambda_j\}$, this is written as:

$$\mathcal{ET}_j(\lambda_j) = \sum_{i,j}^{N,P} \left(T_j(\lambda_j) + T_j^{W2F}(\lambda_j) + T_j^{W2W}(\lambda_j)\right) \tag{3.2}$$

In Step 4, for every parameter in Φ, there is a need to define the boundary values of the $\pi \in \Pi$ under consideration. A key role for deriving appropriate boundary relationships, is played by the possibility that the perturbation parameter is a continuous or a discrete variable. For our analysis, Π has only one parameter: $\pi_1 = \lambda_j$. It is really a matter of taste to consider λ_j a discrete or a continuous variable. Traditionally, the load of a processor for task scheduling in heterogeneous systems is measured either as the number of processes in the processor run-queue, or as the processor delivered speed. The first measure renders λ_j a discrete parameter, taking integer values larger than or equal to 1. The second measure renders λ_j a continuous parameter, measured as availability percentage of the particular processor (in our case m_j) for computing our tasks, usually taking values of 60%, 80% or 95%.

In this work, λ_j is considered a *continuous* variable that measures the availability of processor m_j in %. The percentual availability of a procesor expresses its delivered computational speed, which encompasses all three effects of applications' requirements, hardware capabilities and network speed in one. The boundary values of λ_j must satisfy the following boundary relationships:

$$\left\{\lambda_j \in \left\langle \lambda_j^{'}, \lambda_j^{''} \right\rangle \mid \left(f_1(\lambda_j^{'}) = \beta_1^{\min}\right) \wedge \left(f_1(\lambda_j^{''}) = \beta_1^{\max}\right)\right\} \tag{3.3}$$

The tolerable variation interval for the performance feature of interest, i.e., $\mathcal{ET}_j$, is given by $\beta_1^{\min}, \beta_1^{\max}\rangle$. Even though it is assumed the processor load will be λ_j^{orig}, this value might differ in practice, due to inaccuracies in load estimations or unforeseeable changes in the environment. The tolerable increase in the *actual* finishing time, $\mathcal{ET}_j$, of processor m_j, considering the effects of errors in the estimation of variations of λ_j, cannot exceed $\tau_1(> 1)$ times its estimated value $\mathcal{ET}_j^{orig}$. Then boundary relationships for this analysis are:

$$\left\{\lambda_j \in \left\langle \lambda_j', \lambda_j'' \right\rangle \mid \left(\mathcal{ET}_j(\lambda_j) = \tau_1 \mathcal{ET}_j^{orig}\right) \wedge (1 \leq j \leq P)\right\} \tag{3.4}$$

Now, there is a need to define a robustness radius, which is the *largest* increase in processor load in any direction (for any combination of processor load values) from the assumed value, that does *not* cause any tolerance interval violation for the execution time of all tasks a_i assigned to m_j. To define the robustness radius we need to choose which norm will give us the smallest variation in the system (and ultimately processor) load. A norm is a function that assigns a strictly positive length or size to all vectors in a vector space, other than the zero vector. The choice of a particular norm depends on the DLS algorithm (and target environment), for which it is required to measure the robustness. Another aspect to be considered when choosing the norm is the actual nature of the selected perturbation parameters. According to the nature of λ_j, a more intuitive norm should be used for determining the robustness radius which in this paper is the ℓ_1-norm. A proper definition of ℓ_1-norm can be found in [5]. In general, the robustness radius can be defined using the ℓ_1-norm as follows:

$$r_{DLS}(\mathcal{ET}_j, \lambda_j) = \max \ \|\lambda_j - \lambda_j^{orig}\|_1 \ s.t. \ \mathcal{ET}_j(\lambda_j) = \tau_1 \mathcal{ET}_j^{orig} \tag{3.5}$$

Finally, the robustness metric is the minimum of all robustness radii, assuming more than one performance feature in Φ:

$$\rho_{DLS}(\Phi, \lambda_j) = \min\left(r_{DLS}(\phi_1, \lambda_j)\right) \ \forall \ \phi_i \in \Phi \tag{3.6}$$

And therefore, the robustness metric of the total parallel time T_{PAR} against variations in the total system load would be:

$$\rho_{DLS}(T_{PAR}, \lambda) = \min(\rho_{DLS}(\mathcal{ET}_j, \lambda_j)), 1 \leq j \leq P \tag{3.7}$$

This flexibility metric can be used to determine the impact of system load variations on the performance of the three adaptive DLS techniques, and differentiate them according to their flexibility against variations in the system load. Figure 3.5 illustrates two possible scenarios. The first scenario, describes the situation when three DLS methods perform similarly in terms of performance, with the difference being in the variation in system load that each of them is able to capture. Based on scenario (a) in Figure 3.5, one should choose the DLS method that has the lowest impact on DLS performance and can handle the largest variation of Λ, which in this case is AF. The second scenario, describes the situation when for the same captured variation of system load, the three DLS methods perform differently in terms of computational performance. For this scenario, one should choose the DLS method that has the lowest impact on DLS performance, for a fixed variation interval of Λ, which is again AF.

To measure the Resilience of DLS in the presence of processor failures, both the *number of tasks to be rescheduled*, N^{resch}, and the *total parallel time*, T_{PAR}, given by the DLS algorithm, should be robust. N^{resch} must not exceed $\tau_2\%$ of the total number of tasks N,

and T_{PAR} must not exceed $\tau_3(>1)$ times it's estimated value T_{PAR}^{orig}. Whenever a processor failure occurs, the DLS algorithm must be able to reschedule the tasks that were assigned to the failed processors dynamically. It is also required to reschedule other tasks on the remaining processors if necessary. The following simplifying assumptions are made: (iv) only resources (e.g. network link, processor) associated with worker processors fail, (v) resource failures occur simultaneously, (vi) resource failures are mutually independent, (vii) resource failures are permanent and, (viii) the DLS algorithm has fault-discovery and fault-recovery mechanisms.

The resilience analysis is elaborated below:

In Step 1, in contrast to the first analysis, the set of performance features has two elements in this case: $\Phi = \{\phi_1, \phi_2\}$, where $\phi_1 = N^{resch}$ and $\phi_2 = T_{PAR}$.

In Step 2, in order to identify which processors fail, the approach proposed by Ali et al. in [5] is used here. Thus, let $\mathbf{F} = [\mathbf{f}_1 \mathbf{f}_2 \ldots \mathbf{f}_P]^T$ be the vector containing the live status of all resources, defined as:

$$\mathbf{f}_j = \begin{cases} 1 & \text{iff processor link } m_j \text{ failed} \\ 0 & \text{otherwise} \end{cases} \qquad 1 \leq j \leq P$$

The original value of $\mathbf{F}$ is expressed by $\mathbf{F}^{orig} = [0\ 0 \ldots 0]^T$, indicating that initially all resources are alive. The perturbation parameters set in this case is $\Pi = \{\pi_1\}$, where $\pi_1 = \mathbf{F}$.

In Step 3, in order to determine the impact of Π over Φ, there is a need to determine separately each of the two relationships:

$$\phi_1 = f_{11}(\pi_1) \tag{3.8}$$

$$\phi_2 = f_{21}(\pi_1) \tag{3.9}$$

which relate ϕ_1, and ϕ_2, respectively, to π_1. The number of tasks that need to be rescheduled is directly proportional to the number of processors that fail: N^{resch} increases as more processors fail. Thus, relationship $(9a)$ becomes as $N^{resch} = f_{11}(\mathbf{F})$. In particular,

$$N^{resch}(\mathbf{F}) = N_p^{resch}(\mathbf{F}) + N_{lb}^{resch}(\mathbf{F}) \tag{3.10}$$

where $N_p^{resch}(\mathbf{F})$ is the total number of tasks assigned to the failed resources that need to be rescued (or restarted), and $N_{lb}^{resch}(\mathbf{F})$ is the total number of 'surviving' tasks, assigned to 'surviving' resources, which the failure-recovery mechanism will need to reschedule together with $N_p^{resch}(\mathbf{F})$ with the goal of achieving and then maintaining a good load balancing on the remaining active processors. Additionally, $N_{lb}^{resch}(\mathbf{F})$ also depends on the choice of the DLS algorithm in use. It follows that the total parallel time T_{PAR} increases when the computational resources start to fail. Hence, T_{PAR} is expected to vary with respect to $\mathbf{F}$ and relationship $(9b)$ can be written as $T_{PAR} = f_{21}(\mathbf{F})$. The exact impact of $\mathbf{F}$ over T_{PAR} depends on the choice of DLS algorithm, as well as on its fault-discovery and fault-recovery mechanisms.

In Step 4, there is a need to define the boundary values of $\mathbf{F}$, for which each element in Φ is less than the maximum tolerable number. In this work, $\mathbf{F}$ is assumed to be a *discrete* variable, that measures the number of live resources. Thus there is a need to determine all the pairs of $\mathbf{F}$, such that for a given pair, the boundary value is the one that falls in the robust region. Assume that $\mathbf{F}'$ is a perturbation parameter value, such that the resources that fail in the situation represented by $\mathbf{F}'$, include the resources that fail in the situation

represented by $\mathbf{F}$ and *exactly one* other resource. Then, the boundary relationships can be written as follows, in which T_{PAR}^{orig} is the estimated parallel time assuming that the system is completely safe, i.e., $\mathbf{F}^{orig} = [0\ 0 \ldots 0]^T$.

$$\left\{\mathbf{F} | \left(N^{resch}(\mathbf{F}) \leq \tau_2 N\right) \wedge \left(\exists \mathbf{F}' s.t.\ N^{resch}(\mathbf{F}') > \tau_2 N\right)\right\} \tag{3.11}$$

$$\left\{\mathbf{F} | \left(T_{PAR}(\mathbf{F}) \leq \tau_3 T_{PAR}^{orig}\right) \wedge \left(\exists \mathbf{F}' s.t.\ T_{PAR}(\mathbf{F}') > \tau_3 T_{PAR}^{orig}\right)\right\} \tag{3.12}$$

The robustness radius for this case in a similar manner to the previous case. Given the nature of the perturbation parameter, we use the ℓ_1-norm for determining the robustness radii in a generalized form:

$$r_{DLS}(N^{resch}, \mathbf{F}) = \max \|\mathbf{F} - \mathbf{F}^{orig}\|_1 s.t. N^{resch}(\mathbf{F}') > \tau_2 N) \tag{3.13}$$

$$r_{DLS}(T_{PAR}, \mathbf{F}) = \max \|\mathbf{F} - \mathbf{F}^{orig}\|_1 s.t. T_{PAR}(\mathbf{F}') > \tau_3 T_{PAR}^{orig}) \tag{3.14}$$

$r_{DLS}(N^{resch}, \mathbf{F})$ is the largest number of resources that can fail until N^{resch} exceeds its tolerable value. $r_{DLS}(T_{PAR}, \mathbf{F})$ is the largest number of resources that can fail until the degradation of T_{PAR} is less than $\tau_3 T_{PAR}^{orig}$. An important assumption for this work is that if the system is completely safe and no resource failures occur, then T_{PAR}^{ft} given by the fault-tolerant DLS algorithm should be comparable to the T_{PAR} of the non fault-tolerant DLS algorithm. The robustness metric is the minimum of all robustness radii:

$$\rho_{DLS}(\Phi, \mathbf{F}) = \min\left(r_{DLS}(\phi_j, \mathbf{F})\right)\ \forall\ \phi_i \in \Phi \tag{3.15}$$

3.3.3 Notes on the usefulness of the proposed robustness metrics

The proposed flexibility and resilience robustness metrics depend on certain application, system or algorithm specific parameters, most of which can be determined *a priori*.

If certain parameters become available (or known) only at runtime, the metrics are formulated using initial values (e.g., every element of vector $\mathbf{F}$ is zero or the system load Λ is equal to the original system load Λ^{orig}), which are updated in the master when newer values become available (e.g., vector $\mathbf{F}$ contains non-zero elements or Λ is smaller than Λ^{orig}). In brief, the usefulness of the proposed metrics is twofold: (1) *the metrics can be formulated offline with application, system, and/or algorithm-specific initial values and integrated into the scheduler to guide and adapt autonomously the scheduling decisions*, and (2) usable in conjunction with other desired performance metrics (e.g. makespan) for differentiating among DLS that have similar performance from the makespan-only viewpoint.

The choice of tolerance factors, τ_1, τ_2 and τ_3 can increase or decrease the robustness of DLS algorithms, thus, they should be chosen such that they reflect reality with high accuracy. Table 3.1 gives some suggested values for these parameters.

Table 3.1: Bounds on the tolerance factors, τ_1, τ_2, τ_3, and their suggested average case values

Tolerance factor	**Depends on**	**Best case**	**Worst case**	**Average case**
τ_1	Application type	1.0	1.5	1.25
τ_2	DLS method of choice	0%of N	50%of N	25% of N
τ_3	DLS method of choice, # of failures, fault detection & recovery mechanism	S_p^{ideal}	1	$\frac{S_p}{2}$

The metrics proposed in this work, are essential to bringing the most adaptive and efficient DLS algorithms to the state-of-the-art performance and robustness levels imposed

by today's computing platforms and applications. A careful choice of the tolerance factors makes the proposed metrics useful towards producing efficient, qualitative and reliable schedules for execution of large and complex scientific applications.

3.4 A Combined Dual-stage Framework for Robust Scheduling of Scientific Applications in Heterogeneous Environments with Uncertain Availability

Studies for determining the best techniques to be used for each stage (of a two-stage framework) that: (a) maximize the probability that the system makespan satisfies a deadline in stage I of initial static mapping, and (b) minimize the system makespan for every given availability level in the system in stage II of pplication scheduling at a fine grain level, are presented in this section [32].

3.4.1 Motivation

Using robust resource allocation (RA) heuristics [6] *and* application load balancing via dynamic loop scheduling (DLS) techniques, in concert, will enhance the execution of computationally intensive scientific applications in uncertain heterogeneous systems. The goal of this research is to assign applications to heterogeneous computing systems and execute them in such a way that all applications complete before a common deadline, and their completion times are robust against uncertainty in input data *and* system availability.

To accomplish this goal, the approach proposed herein is to divide the execution of scientific applications on heterogeneous computing systems into two stages, as outlined in Figure 3.6: Stage I *initial mapping*–resources are allocated to each application according to a given robust RA policy, and Stage II *runtime application scheduling*–the execution

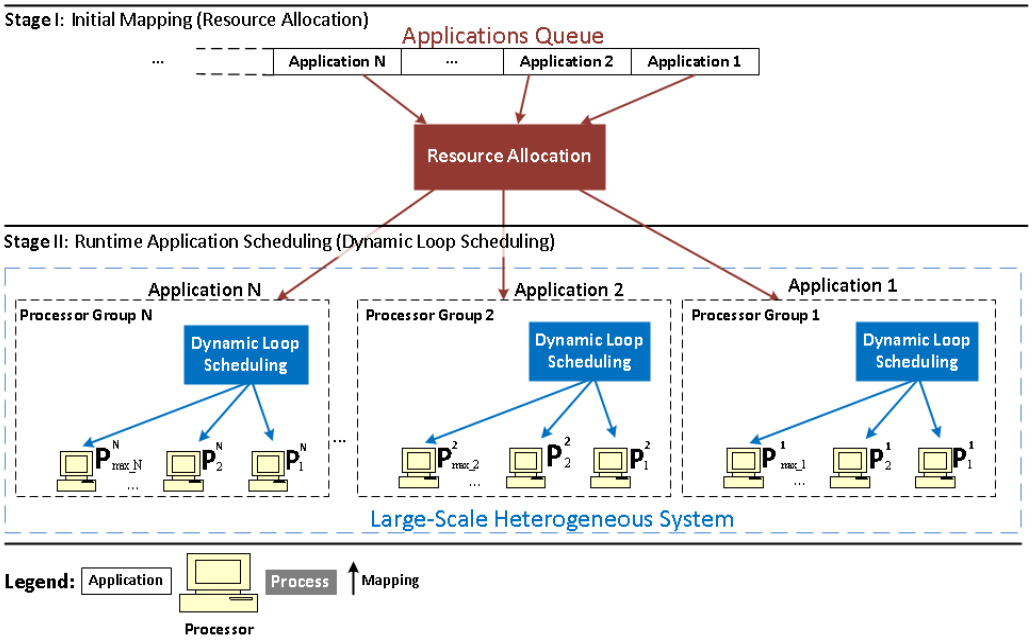

Figure 3.6: Schematic illustration of the proposed dual-stage framework.

of each application is optimized, for the set of resources allocated in the previous stage, according to a given robust application scheduling strategy.

Initial mapping (IM) can be defined as the problem of finding a mapping of a batch of applications onto a set of resources to maximize robustness against uncertain input data and system availability. Robustness here is defined as the probability that applications are completed on the allocated resources by a common deadline [99].

The motivation for solving the IM problem via robust RA is to avoid the runtime resource reallocation problem, i.e., reallocating resources already assigned to applications to avoid violations of the performance objective. The robustness of an RA can be quantified as the joint probability that all applications will complete by their deadline given the uncertain input data and system availability.

Just as in stage I, uncertain runtime availability of resources allocated to an application, as well as uncertain input data, are known sources of uncertainty in stage II and may impact the applications execution times. The motivation for this stage is based on the assumption that a specific runtime application scheduling (RAS) policy exists that avoids the runtime resource reallocation problem and that satisfies the stated performance objective, while possibly allowing a larger degree of uncertainty in input data and system availability.

3.4.2 Outline of the combined dual-stage framework

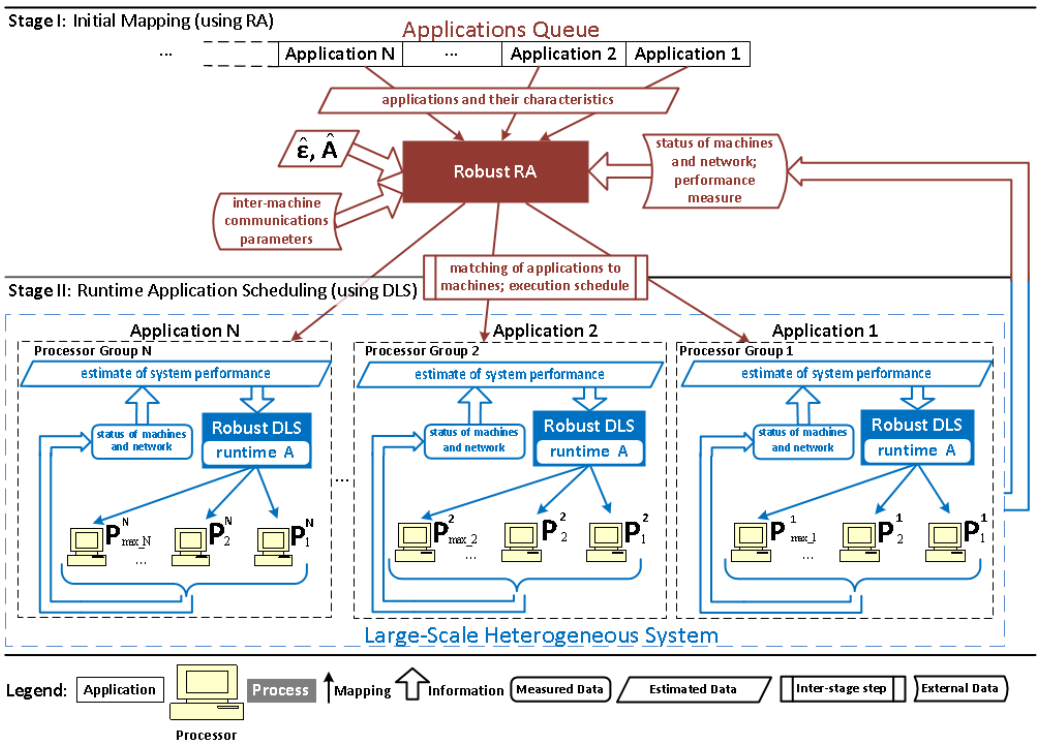

Figure 3.7: Schematic illustration of the proposed combined dual-stage framework with robustness.

The proposed CDSF for robust execution of scientific applications on heterogeneous uncertain computing systems is schematically illustrated in Figure 3.7.

Initial mapping conducted in stage I is the problem of finding a static mapping (i.e., one found in an offline planning phase) of a batch of applications onto a set of resources to maximize robustness of the allocation against uncertain input data and system availability, by maximizing the probability that all applications will complete before the deadline, given a probability mass function (PMF) for system availability $\hat{\mathbf{A}}$. Runtime application scheduling carried out in stage II is the problem of finding a dynamic scheduling policy for each application that minimizes the parallel time to complete of an application for every given runtime system availability $\mathbf{A}$.

In *Stage I – initial mapping*, scientific applications arrive at random intervals in the queue of a resource manager, in view of assignment for execution onto any one of a group of resources of a heterogeneous computing system. The applications queue consists of different scientific applications, which can represent different instances of the same application. As the applications arrive, their assignment to available resources is made in batches. After assignment, an application is placed in the input queue of the resource designated as coordinator (master) of the assigned group of resources. Any required data are staged at the master, in advance of application execution. Let N be the number of applications in the batch. Each application is assumed to be *data parallel* (with no interprocessor communications needed) and to contain large computationally intensive parallel loops. Robust heuristics are employed for the initial mapping, and the intention is to conduct studies to determine the best heuristic to use in this stage. The best heuristic will provide the most ro-

bust mapping of groups of resources to applications, i.e., maximize the probability that an application completes before $\mathbf{\Delta}$, assuming a certain system availability $\hat{\mathbf{A}}$. The resource allocation actions are pre-planned before the actual execution of the applications begins and the goal is to minimize (or to prevent) the immediate effects of uncertain perturbation in $\hat{\epsilon}$ and $\hat{\mathbf{A}}$ on the system makespan $\mathbf{\Psi}$, such that $\phi_1 = \{\mathbf{Pr}(\mathbf{\Psi} \leq \mathbf{\Delta})\}$ is maximized. Regardless of the type of allocated resources, once an allocation decision has been made, it cannot be adjusted for a currently executing application. Perturbations during the actual execution of applications are expected and addressed (or compensated for) in stage II via the use of robust DLS techniques. Let max_i, $i = 1, N$ be the number of resources allocated to application i, and $T^{exp}_{max_i,i}$ be the expected time to complete of application i on max_i processors.

In *Stage II – runtime application scheduling*, each application from the current batch of N applications is executed on its group of resources allocated in stage I. A robust DLS technique from the set {FAC, WF, AWF-B, AF} [15][107][13] is employed to define the rules for executing an application at runtime. The intention is to conduct studies to determine the best DLS technique to employ for each application in the batch, such that the completion time of an application is minimized for every given runtime system availability $\mathbf{A}$, and consequently, the system makespan is smaller than or equal to the deadline. A single DLS technique may be employed for several applications as several distinct instances of the particular DLS technique. In general, the runtime system availability is expected to be different than the estimated system availability. In this work, it is assumed that $\mathbf{A} \leq \mathbf{E}[\hat{\mathbf{A}}]$. The most robust DLS technique will provide the best runtime scheduling decisions for

executing an application on the allocated group of processors that minimize the system makespan while tolerating a larger degree of perturbation in system availability than the one assumed in stage I. The goal of the robust DLS technique is to detect any runtime perturbation in system availability as soon as it occurs, and to take appropriate scheduling decisions for the remaining unexecuted application iterations. Stage II can, thus, be considered a runtime approach for the detection and recovery from the uncertain effects of the perturbation expected to occur in $\mathbf{A}$, on the performance feature for this stage. To guide the scheduling decisions at runtime and to tune the performance of an application, the DLS techniques use runtime estimations of the time required to compute loop iterations. These times are determined using probabilistic analyses and are influenced by the application input data and the availability to compute of the resource executing the iteration(s). The execution of an application using a DLS technique is non-preemptive, and, therefore, the choice of the DLS technique cannot be changed during runtime. The overhead associated with employing a robust DLS technique is higher than that of a robust RA heuristic. The actions are not pre-planned and are taken dynamically during the application execution, as soon as perturbation occurs. The benefits are expected to, and in general do, compensate the overhead of employing robust DLS techniques. Let ρ_1 be the largest robustness value of stage I. Also, let ρ_2 be the largest robustness value of stage II. The system robustness is quantified as the 2-tuple (ρ_1, ρ_2).

3.4.3 Usefulness of Proposed Framework

The assessment of the usefulness of the proposed CDSF requires an investigation of the impact of the different possible RA heuristics and DLS techniques on the performance objective of interest. A small scale example was provided illustrate the usefulness of the proposed CDSF in [32]. The data that was chosen for this example was used to demonstrate the efficacy of the CDSF. A heterogeneous system with twelve processors of two types was considered, where processors of type 1 are assumed to have a different computational capacity and availability than processors of type 2. A batch of $N = 3$ applications is considered, having different sizes and serial/parallel component ratios. Serial iterations can only be executed on a single processor and parallel iterations can be executed on multiple processors of the same type. The system deadline considered was $\boldsymbol{\Delta} = 3,250$ time units, and was chosen to help illustrate the differences between using intelligent stages (stages that consider robustness at both stages) versus naïve stages (stages that do not consider robustness at either stage) in the dual-stage framework. The usefulness of the proposed CDSF is based on the hypothesis that any of the naïve scenarios will result in solutions that tolerate much *less* perturbations variations in the overall system, and therefore, are *less* robust. Thus, the scenario (robust IM$-$robust RAS) is expected to be superior to the other scenarios. The CDSF allows investigation of the overall degree of tolerable uncertainty for which the stated performance objective is satisfied, at the level of each individual application and for the entire batch of applications executing on the heterogeneous system.

3.5 Analyzing the Robustness of Dynamic Loop Scheduling for Heterogeneous Computing Systems

In this work, a methodology is proposed for performing robustness analysis of the dynamic loop scheduling techniques using a metric, formulated in earlier work, to measure their robustness in heterogeneous computing systems with uncertainties. The dynamic loop scheduling methods have been implemented in a simulation. The experimental results were used as an input to the proposed methodology, which in turn has been used to experimentally analyze the robustness of a number of dynamic loop scheduling methods on a heterogeneous system with variable processor availability [110].

3.5.1 Motivation

The theoretical foundation of measuring the robustness value of a set of DLS methods with respect to variation in the processor loads has been described in earlier work [15][107]. Further, an experimental analysis of the robustness of the DLS methods is required to compare and select the most robust DLS technique to achieve an optimal execution performance in the presence of perturbations in processor loads. The central idea is to simulate the execution of data parallel loops within a scientific application exhibiting irregular behavior, and to simulate the uncertainty in the variation of processor weights using random probability distributions. The simulation results are used to experimentally analyze and evaluate the robustness of the DLS methods for various execution scenarios, which reflect the performance of the DLS methods in both dedicated and non-dedicated computing environments. To exemplify the execution environments, a set of tolerable threshold values is chosen, and the robustness of the DLS methods is measured (using the robustness metric

described in [15] and [107]) against the variation of processor weights with respect to these threshold values.

3.5.2 Experimental Analysis and Evaluation

The *robustness* value of the scheduling method, given by r_{method}, is defined as the increase in the value of T_{PAR} from its ideal expected value, T_{PAR}^{ideal}, for a fixed variation in the system load. The *robustness metric*, ρ, is defined as the minimum of all robustness radii values. The actual parallel execution time T_{PAR} obtained in the presence of variation in the system load must not exceed by τ (where $\tau > 1$) times the ideal expected execution time for the application, T_{PAR}^{ideal}. This can be mathematically expressed as:

$$T_{PAR} \leq \tau \cdot T_{PAR}^{ideal} \tag{3.16}$$

To measure the robustness of the DLS methods for non-dedicated heterogeneous systems we examine two cases that have been generated for an experimental study of robustness. The application being considered for the experimental simulation study has N loop iterations and the iteration execution times follow a Gaussian distribution as shown in Figure 3.8. The first case is generated to obtain the value of T_{PAR}^{ideal} for the application on a set of P heterogeneous processors, with constant processor availability or no processor weight variation. In the second case, T_{PAR} is measured for the same application in the presence of varying processor weights. In this case, the Gamma distribution is used to represent the variation in processor weights over time, as shown in Figure 3.9. The figure illustrates a total of 32329 processor weight samples distributed over 60 weight intervals for a system of upto 4096 processors. The robustness analysis conducted in this paper can be applied

to any type of distribution. However, for the purpose of illustration, we are reporting the results obtained using Gaussian distribution for iteration execution times and Gamma distribution for variation in processor weights. The DLS techniques have been implemented in a simple simulation using priority queues. The experimental results obtained via the use of this simulation confirmed the expectations raised by the earlier analytical studies [13][27]. The robustness analysis described in this paper using the simulation results is a preliminary step towards a more comprehensive analysis of the robustness of the DLS methods. In this work, the scientific application being executed is considered to contain N independent loop iterations and the execution times of the loop iterations are represented using a Gaussian distribution as shown in Figure 3.8. The loop iterations are generated in a separate file, which is provided as an input to the simulation code implementing the DLS methods.

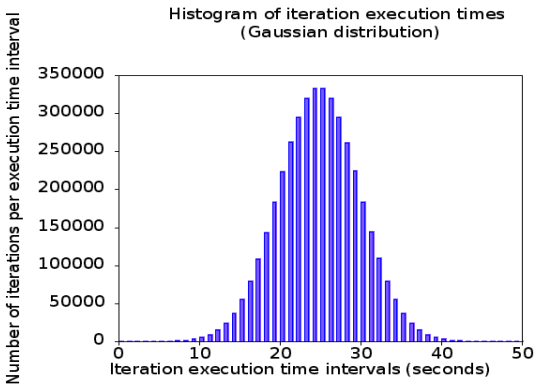

Figure 3.8: Iteration execution times generated using Gaussian distribution with $\mu = 25$ and $\sigma = 5$.

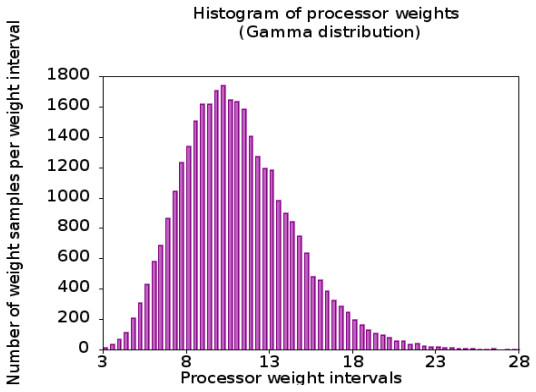

Figure 3.9: Variation in processor weights as a Gamma distribution with $\mu = 11$.

The computing system is modeled as a system with P heterogeneous processors. The simulation of the system is divided into two cases as mentioned earlier:

(i) the system is comprised of processors with constant weights, and

(ii) the system is comprised of processors with varying weights.

For a system with constant processor weights, the processors are assigned a fixed weight value at the beginning of the simulation. These weights do not change for the entire duration of the simulation. For a system with varying processor weights, Gamma distribution is used to characterize the variation in the processor weight values, as shown in Figure 3.9. In both cases, the processor weight values are read from a separate input file during the simulation of the execution of the DLS methods. Also, in both cases, the experiments have been conducted for all the DLS methods, namely, FSC, GSS, FAC, WF, AWF-B, AWF-C, and AF, and for the straight forward parallelization, STATIC. A number of experiments have been performed for different values of N and P. Figures 3.10 to 3.15 show the performance of the above DLS techniques on computing systems with constant and varying

processor weights, respectively. The longest time required to conduct one simulation test (i.e., simulation time) for $N = 4194304$ iterations and $P = 4096$ simulated processors for all the DLS methods was 19 hours on a physical computer with an Intel(R) Core(TM) i5-2430M processor and 4.00 GB RAM.

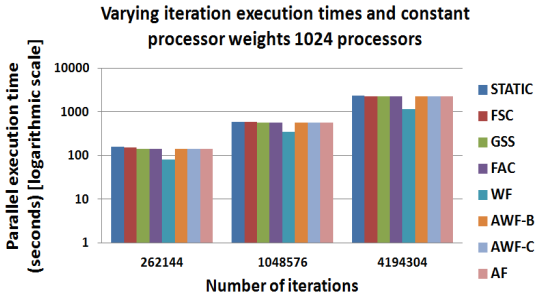

Figure 3.10: Execution of the DLS methods and STATIC on a system with 1024 processors and constant processor weights equal to 27.66

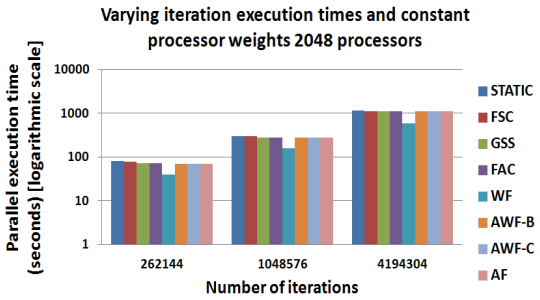

Figure 3.11: Execution of the DLS methods and STATIC on a system with 2048 processors and constant processor weights equal to 27.66

The behavior of the DLS methods for case (i) is illustrated in Figures 3.10 to 3.12. The processors are initially assigned a constant weight value at the beginning of the simula-

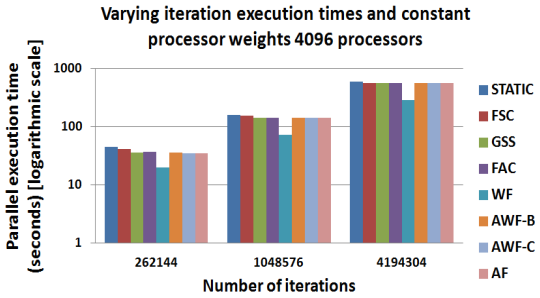

Figure 3.12: Execution of the DLS methods and STATIC on a system with 4096 processors and constant processor weights equal to 27.66

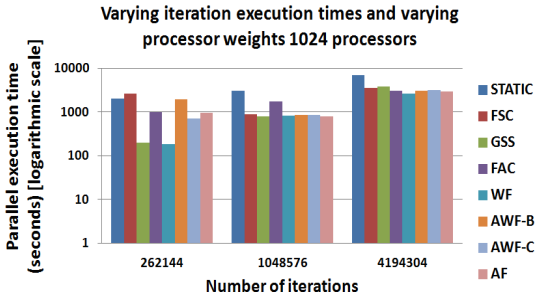

Figure 3.13: Execution of the DLS methods and STATIC on a system with 1024 processors and varying processor weights in the [3.52, 27.66] range

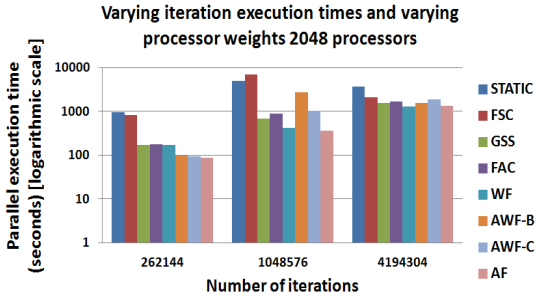

Figure 3.14: Execution of the DLS methods and STATIC on a system with 2048 processors and varying processor weights in the [3.52, 27.66] range

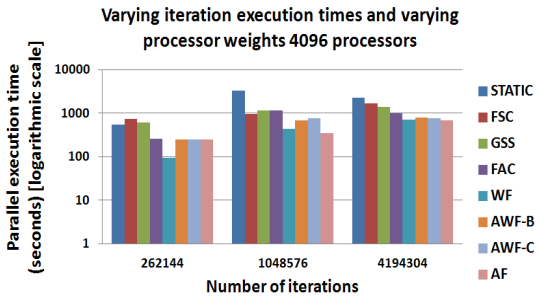

Figure 3.15: Execution of the DLS methods and STATIC on a system with 4096 processors and varying processor weights in the [3.52, 27.66] range

tion, which does not change during the execution of the application. For the experiments shown in this paper for case (i) the processors have been assigned a weight value of 27.66. Given that the availability for the processor with the minimum weight value of 3.52 has an availability of 12.73% and the processor with maximum weight value of 27.66 has an availability of 100%, the availability for the processor weights considered for case (i), which is also referred to as an ideal case for executing the scientific applications, uses processors weights with maximum availability. Recall that, the iteration execution times follow the Gaussian distribution shown in Figure 3.8. This variation in iteration execution times was the same for each DLS method, to simulate the execution of the same application using every DLS technique. Figure 3.8 shows that the iteration execution times are distributed over a defined range of values $[0.1361s, 49.9082s]$. However, the processor weights remain constant throughout the execution of the entire application, which causes admissible load imbalance in the execution of the application. In this case, the non-adaptive DLS methods are sufficient to provide an optimal performance. For instance, in Figures 3.10 to 3.12, WF outperforms all other DLS techniques by almost a factor of 2. The superior performance

of WF can be explained by the fact that this technique has a knowledge of the processor weights before the first scheduling step and that in this case, the processor weights do not change over time. Thus, WF is capable of scheduling optimal sized chunks to each processor at every scheduling step.

The behavior of the DLS methods for case (ii) is illustrated in Figures 3.13 to 3.15. The iteration execution times follow the same distribution as in case (i). However, in this case, the processor weights vary randomly over time according to the Gamma distribution as shown in Figure 3.9. Figures 3.13 to 3.15 show a general trend that the adaptive DLS techniques outperform the non-adaptive DLS techniques, and among the adaptive ones, AF performs the best. This is due to the fact that AF allows relaxation of some of the theoretical assumptions imposed by the models used in the other DLS methods, which make AF be inherently more robust to the extreme load variation caused by the random variation in the processor weights. In general, the number of chunks produced by an adaptive DLS technique is greater than the number of chunks produced by a non-adaptive DLS technique. For any variation in the processor weights, a non-adaptive DLS either takes into account the initial state of the system in the first scheduling step, or assumes the processor weights are the same or equal. On the contrary, an adaptive DLS technique determines the processor weights for every chunk size calculation. Thus, the adaptive DLS methods, assign fewer iterations to a more loaded processor having a lower weight value, and more iterations to a less loaded processor with higher weight value. Ideally, in this scenario, the adaptive DLS method is expected to deliver a more load balanced allocation of iterations and a lower application execution time.

Table 3.2: T_{PAR}^{ideal}, T_{PAR}, and $r_{method} = T_{PAR} - T_{PAR}^{ideal}$ values for $N = 1048576$ iterations and $P = 4096$ processors

Method	$\mathbf{T_{PAR}^{ideal}}$ **(s)**	$\mathbf{T_{PAR}}$ **(s)**	$\mathbf{r_{method}}$ **(s)**
STATIC	593.206	2299	1705.794
FSC	562.943	1651.71	1088.767
GSS	561.327	1382.58	821.25
FAC	561.4	982.474	421.074
WF	282.964	707.939	424.975
AWF-B	560.14	786.64	226.5
AWF-C	559.781	762.25	202.469
AF	559.744	689.82	130.076

For the robustness analysis of the DLS methods against the variations in the system load, the results from the test case shown in Figure 3.15 are used, where $N = 4194304$ iterations and $P = 4096$ processors. This particular test case is chosen, as it is executed for the largest number of loop iterations and processors used in our experiment, such that it accounts for a significant amount of load imbalance required to analyze the behavior of the DLS methods. The value of T_{PAR}^{ideal} for each DLS method is calculated from a similar test scenario without the processor weight variation as shown in Figure 3.12. Table 3.2 shows the values of T_{PAR}^{ideal}, T_{PAR}, and r_{method} for each of the DLS methods for the test case where $N = 4194304$ iterations and $P = 4096$ processors. To analyze the robustness of the DLS methods three values of τ are selected, which are 1.25 in the best case, 1.5 in the average case, and 2.5 in the worst case. For the best case when $\tau = 1$, only AF proves to be a robust DLS, and to satisfy the condition given in equation (1). The robustness value of AF is given by r_{AF} equals 130.076 seconds. Similarly, for the average case when $\tau = 1.5$, only AF ($r_{AF} = 130.076$ seconds), AWF-B ($r_{AWF-B} = 226.5$ seconds), and AWF-C ($r_{AWF-C} = 202.469$ seconds) satisfy the condition for robustness. For the worst

case when τ is further relaxed to a value of 2.5, all the scheduling techniques except the DLS methods, GSS ($r_{GSS} = 821.253$ seconds) and FSC ($r_{FSC} = 1088.767$ seconds), and the straight forward parallelization method, STATIC ($r_{STATIC} = 1705.794$ seconds), satisfy the condition for robustness against a fixed variation in the system load. The value of ρ is calculated as the minimum of all the r_{method} values obtained for all the scheduling methods for all values of τ. In this test case $r_{AF} = 130.076$ seconds is the minimum of all values for the robustness of the scheduling techniques. Thus, the value of ρ is 130.076 seconds for this scenario and the robustness values of the DLS methods and STATIC in terms of this robustness metric are as follows: AF $= \rho$, AWF-C $= 1.56\rho$, AWF-B $= 1.74\rho$, WF $= 3.27\rho$, FAC $= 3.24\rho$, GSS $= 6.3\rho$, FSC $= 8.37\rho$, and STATIC $= 13.11\rho$. Thus, the scheduling methods can be arranged in the decreasing order of their degree of robustness against variation in processor weights for this test scenario: AF $<$ AWF-C $<$ AWF-B $<$ FAC $<$ WF $<$ GSS $<$ FSC $<$ STATIC.

3.5.3 Benefits of the Proposed Methodology

Prior work on the performance analysis of the DLS methods has only compared the performance of the DLS methods with respect to the parallel execution time of the application obtained upon employing the DLS methods. In this work, a comparison of the robustness of the DLS methods has been performed. The robustness analysis of the DLS methods tests their *flexibility* toards the variation of system load at runtime. The methodology proposed in this work establishes a general simulation to analyze the robustness of the DLS methods when they are used to execute scientific applications on other types of

systems. The simulation framework provides a more controlled environment for a comparative analysis of the robustness of the DLS methods. To the best of our knowledge, this is the first implementation of a methodology to assess the *robustness* of the DLS methods via simulation. The robustness metric obtained as a result of using the proposed methodology is used to quantify the robustness of the DLS methods, and to compare the DLS methods with respect to their robustness values. The robustness metric can also be used in conjunction with other performance metrics, such as makespan, to describe, for instance the quality of performance of the DLS methods. When used in conjunction with other performance metrics, the robustness metric proves useful when selecting the most robust DLS technique which also yields performance from the point of view of the parallel execution time and cost.

3.6 Predicting the Flexibility of Dynamic Loop Scheduling Using an Artificial Neural Network

In this work, an artificial neural network (ANN) model [97] is proposed to predict the *flexibility* (or robustness against system load fluctuations in heterogeneous computing systems) of dynamic loop scheduling (DLS) methods. The multilayer perceptron (MLP) ANN model [97] has been used to predict the degree of robustness of a DLS method, given specific values for the problem size, the system size, and the characteristics of the system load fluctuations as a compound effect of the variations in the application's iteration execution times and the processor availabilities. The developed MLP ANN model can be useful in an effective selection of the most robust DLS technique for scheduling a certain type of sci-

entific application onto a given set of non-dedicated heterogeneous processors, when their system load is expected to fluctuate unpredictably during the application's runtime [109].

3.6.1 Motivation

The system load is defined as the compound effect of the variations in the problem characteristics (loop iteration execution times) and the systemic characteristics (processor availabilities). In the previous work, the robustness of the DLS methods was only assessed with respect to the variations in the processor availabilities for a manageable number of test cases [110]. However, it is of interest to investigate the robustness of dynamic loop scheduling via an ANN analysis, to ensure the robustness of the DLS methods for a considerably larger number of experimental cases resulting from considering different combinations of problem sizes, number of processors, and scheduling methods. Forecasting robustness using traditional statistical specifications in the form of tables and equations for performing a comprehensive manual robustness analysis is a very time consuming and tedious task. Traditional statistical approaches often assume a linear functional relation between the actual parallel execution performance feature and the affecting perturbation parameters. However, the actual performance shows highly non-linear behavior in relation to the perturbation parameters. A multilayered perceptron (MLP) ANN model is used in this paper to predict the *flexibility* of the DLS methods in the presence of system load fluctuations, by learning the relation among the following attributes: loop scheduling methods, problem sizes, system sizes, characteristics of the system load fluctuations as a compound effect of the characteristics of the variations in the iteration execution times and the pro-

cessor availabilities, and impact of system load fluctuations on the parallel execution time. The MLP ANN model generates a non-linear function from the perturbation parameters (i.e., system load) to the performance feature (i.e., execution time). The ANN does not require any prior assumption of the type of functional relation between the above mentioned attributes. The ANN is trained on a part of the input data set and tested with a different subset of the input data set. The results for generating the input dataset have been obtained in a similar manner as described in the work presented in [12]. Furthermore, when exposed to new data, the MLP ANN is capable of accurately predicting the DLS robustness. This work is a proof of concept that ANNs can be employed to predict the *flexibility* of DLS in the presence of fluctuating system load. Preliminary results obtained in [12], have been used to verify this concept. The results obtained from the work done here provide a novel contribution towards predicting the *flexibility* of scheduling in parallel and distributed computing, considering that very limited work has been done in this area [116][41].

3.6.2 Design of the MLP ANN Model

In this work, an MLP ANN model is developed to predict the flexibility of the loop scheduling methods in the presence of system load fluctuations. The model is generated using the MLP classifier of the open source data mining tool, Weka [55]. The input data set for the proposed MLP consists of parallel execution times obtained using a simulation toolkit as described in [12] and using the robustness analysis methodology proposed in [110]. These execution time values have been obtained for various problem sizes, system sizes and probability distributions characterizing the fluctuations in system load. The

degree of robustness is defined in the MLP input data set as the range of values [1, ... 5], where 5 is the highest degree of robustness denoting the most robust scheduling method for a particular execution scenario, and 1 denotes a non-robust scheduling method. For a given number of processors, a given number of loop iterations, and a particular probability distribution for the variations in the iteration execution times and the processor availabilities, the degree of robustness is calculated as follows:

- Calculate the parallel execution time for the ideal case T_{PAR}^{ideal}, for each scheduling method, where the computing system has dedicated processors with 100% availability, for each scheduling method.
- Calculate the parallel execution time T_{PAR}, where the application has variable iteration execution times, and the computing system has non-dedicated processors with variable availability.
- Set the values of the tolerance factor, τ, enforcing an upper limit to the impact of fluctuating system load on T_{PAR}, as 1, 1.25, 1.5, and 1.75.
- A scheduling method is robust if it satisfies the condition $T_{PAR} \leq \tau \cdot T_{PAR}^{ideal}$. Thus, the degree of robustness is 5 for $\tau = 1$, 4 for $\tau = 1.25$, 3 for $\tau = 1.5$, 2 for $\tau = 1.75$, and 1 for all values of $\tau > 1.75$.

For the calculation of the degrees of robustness, the values of T_{PAR}^{ideal} and T_{PAR}, were obtained for all possible combinations of the values of the following parameters: *schedulingMethod* $= \{$STATIC, FSC, GSS, FAC, WF, AWF-B, AWF-C, AF$\}$, $P = \{2048, 4096, 8092\}$, $N = \{1048576, 4194304, 16777216\}$, *iterationDistribution* $= \{Gaussian, Gamma, Exponential\}$ and *availDistribution* $= \{Uniform, Exponential\text{-}constant, Exponential\text{-}variable\}$. The final input dataset for our experiment contained 1,152 samples (or instances). The input dataset is preprocessed using the *NumericToNominal* preprocessor in Weka, which converts all of the numeric values, such as $8,092$ and $4,194,304$, into categorical values. Thus, during the ANN learning, these values represent categories. This step

prevents the magnitude of N from dominating all of the calculations. We then converted each instance into a 22-dimension binary vector with the *NumericToBinary* preprocessor in Weka. In this step, a binary variable is introduced for each value of each parameter. For example, there is a binary variable corresponding to $availDistribution = Gaussian$. Each instance is converted to a binary vector by setting the binary variables corresponding to the parameter values for that instance to 1 and the others to 0. We perform this transformation to avoid suggesting an ordering to the parameter values. For example, if for $availDistribution$, we encoded $Gaussian = 1$, $Gamma = 2$ and $Exponential = 3$, then that implies to the learning algorithm that $Gamma$ is "closer" to $Gaussian$ than $Exponential$. The binary encoding avoids this implication. The $degreeOfRobustness$ was similarly transformed. In order to minimize any bias in the distribution of samples, due to the skewed nature of the collected data, during our cross-validation evaluation, we randomized the ordering of the instances. Different number of iterations of training were used during backpropagation. However, the results reported in this paper have been generated with 500 iterations of training (or epochs). This offers a good tradeoff between the computing time required to train the MLP and overfitting the training set. Empirically, other numbers of iterations resulted in similar prediction accuracy and increased or decreased the computation time linearly. The preprocessed data is then fed as the input dataset to the MLP classifier in Weka. We use stratified 10-fold cross-validation to divide the dataset into the respective training and testing sets. The goal of the MLP is to predict the degree of robustness given scheduling method, P, N, iteration distribution and availability distribution, and as such we select degree of robustness as the *class variable*. During training, we

use the class variable to adjust the edge weights. Backpropagation is then used to train the MLP. After training for each cross-validation fold, we use the MLP to predict the degree of robustness for the test data set. While testing, the class variable is treated as unknown and predicted with the MLP. Based on the predictions, we calculate the accuracy and balanced error rate of the model.

3.6.3 Experimental Analysis and Evaluation

The MLP ANN model generated using Weka is used to predict the *flexibility* of the scheduling methods as a measure of their degree of robustness against system load fluctuations in a given execution scenario. The structure of the MLP ANN, generated using Weka, is illustrated in Figure 3.16. The MLP ANN has an input layer with 22 neurons, one hidden layer with 13 neurons, and one output layer with 5 neurons.

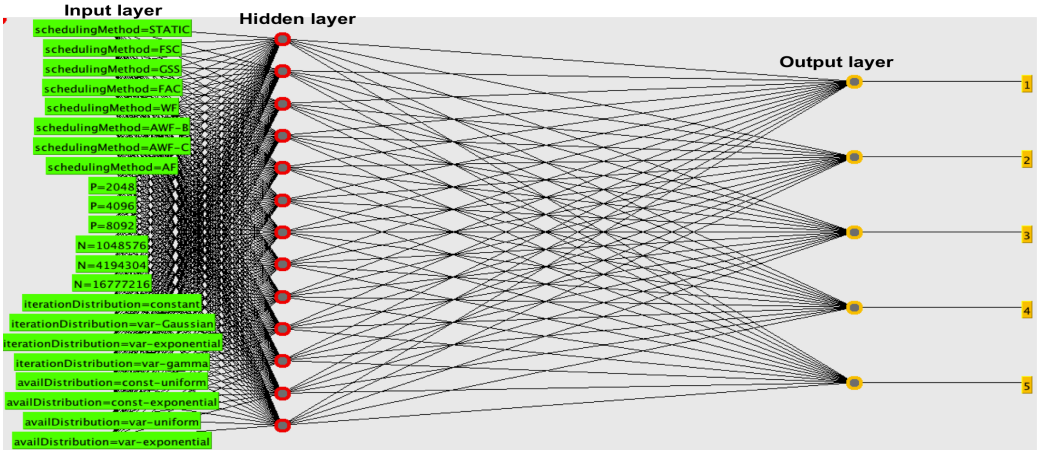

Figure 3.16: Weka-generated MLP ANN with five input attributes and one output class attribute.

The accuracy of the MLP ANN for predicting the degree of robustness was 0.95 on the test dataset samples, which were not used during training. The balanced error rate of the MLP ANN was 0.58. The time taken by Weka to build and train the MLP ANN model was 9.55 seconds, and the time taken to test and validate the ANN model was 0.94 seconds. These timings were recorded by Weka on an Apple® computer with an Intel® Core i5-2.3GHz processor and 4.00 GB RAM.

For comparison, we show the prediction errors of the learned MLP ANN compared to the errors of a simple 0-R classifier, which simply predicts the most common value (robustness class 1, in this case) for all samples. As expected, and confirmed in Figure 3.17, the MLP ANN outperforms the naive 0-R classifier. Of course, the 0-R classifier correctly predicts all of the samples for class 1, but does not distinguish between any of the other four robustness classes. For comparison, this gives the 0-R classifier a seemingly impressive accuracy of 0.90 (5% worse than the MLP ANN, though); however, its balanced error rate was 0.8. These results call attention to the skewed distribution of our data and highlight the difficulty and importance of distinguishing between robustness classes.

Qualitatively, as shown in the confusion matrices in Figure 3.17, the MLP ANN only mis-classifies 10 instances (out of 62) from class 5 and correctly identifies 8 instances (out of 31) from class 2. These correct predictions are important for downstream planning because they allow a scheduler to effectively decide which loop scheduling algorithm will best adhere to user constraints. In contrast, the incorrect predictions by the 0-R classifier could lead to very poor decisions about which scheduling algorithm to prefer. Both the MLP ANN and the 0-R classifier were unable to correctly predict any instances from

classes 3 or 4. The MLP ANN was unable to distinguish these classes because of the limited number of training instances available for them (13 and 7, respectively). A larger training set would allow the backpropagation algorithm to better learn the characteristics of these classes. The use of a larger training dataset is potential future work to the preliminary work done in this paper for predicting the flexibility of DLS with ANNs.

A scatter plot of the predicted values of the degree of robustness for all scheduling methods is in Figure 3.18. Furthermore, the visual interface of the output result in Weka allows a detailed view of any coordinate point on the scatter plot in a textual mode. This visual interface is shown in Figure 3.18 as the two overlaid GUI windows (one showing correctly predicted degree of robustness for AF, and the other showing incorrectly predicted degree of robustness for STATIC) on top of the scatter plot. For a desired degree of robustness and a particular scheduling method, this detailed view helps in identifying the required execution scenario. Similarly, the visual output of the ANN model in Weka enables the identification of the required values of all input parameters in the execution scenario, when the output class attribute (degree of robustness) is plotted against against any other input attribute. For example, this can help in selecting the most flexible scheduling method to achieve a desired level of robustness, for a given problem size, system size and assumed variations in the system load characteristics. Additionally, the MLPs learned by Weka can be exported and used online. For a given execution scenario, they can be used to predict the degree of robustness of each scheduling method in real time. Then, the scheduling method predicted to be the most robust can be selected to run the desired job.

MLP Confusion Matrix

Actual Class \ Predicted Class	1	2	3	4	5
1	1033	4	0	0	2
2	19	8	3	0	1
3	9	1	0	1	2
4	6	1	0	0	0
5	9	2	0	0	52

(a)

0-R Confusion Matrix

Actual Class \ Predicted Class	1	2	3	4	5
1	1039	0	0	0	0
2	31	0	0	0	0
3	13	0	0	0	0
4	7	0	0	0	0
5	62	0	0	0	0

(b)

Figure 3.17: The confusion matrices of (a) the MLP ANN and (b) the 0-R classifier.

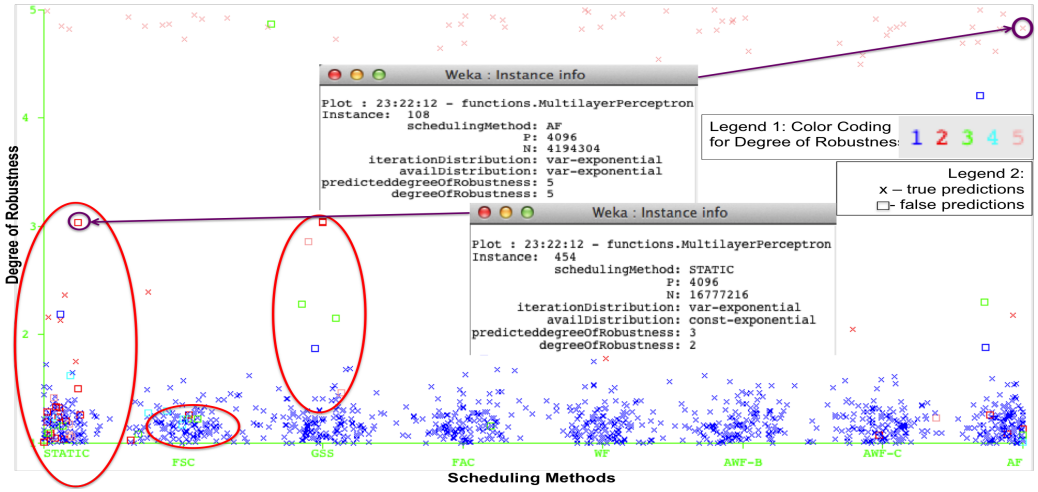

Figure 3.18: Degree of robustness predictions obtained from the MLP ANN model.

As illustrated by the areas highlighted via the red circles in Figure 3.18, the STATIC, FSC and GSS scheduling methods contribute towards the largest number of incorrectly predicted degree of robustness values by the MLP ANN. The statistical results generated by Weka show that STATIC, FSC and GSS together contribute to 79% of the total number of incorrectly predicted values by the proposed MLP ANN model. This indicates that STATIC, FSC and GSS are less predictable compared to the other scheduling methods because, even though the MLP can model highly non-linear functions, it was still unable to consistently predict the robustness of these three methods. All results are in confirmation

with the theoretical and the experimental results obtained for these scheduling techniques in related previous work [13][110].

3.6.4 Benefits of the MLP ANN Flexibility Prediction Model

In previous work, a methodology to assess the robustness of the scheduling methods has been proposed to test their robustness against variations in processor availabilities at runtime. The methodology proposed in [110] establishes a general procedure to analyze the robustness of the dynamic loop scheduling (DLS) methods when they are used to execute scientific applications on non-dedicated heterogeneous computing systems. The robustness metric is used to quantify the robustness of the DLS methods, and to compare them with respect to their robustness values; however, the work presented in [110] is only a preliminary step towards analyzing the robustness of the the scheduling methods. The statistical calculations in that work are restricted to smaller test data sets.

In this work, an MLP ANN model is used to predict the robustness of the scheduling methods against fluctuating system load and captures more realistic execution scenarios. The advantages of using an ANN model over traditional statistical methods for predicting the *flexibility* of the scheduling methods are: (i) a capability to handle larger input datasets, (ii) a high real time prediction speed (approximately 1 second) as jobs are presented to a computing system, (iii) a capability to learn non-linear relations among input and class attributes (does not require any prior information related to the functional relation among the input and output attributes), and (iv) a capability to make correct predictions of robustness

(of DLS) on data unexplored during training (verified by using 10-fold cross validation technique).

The MLP ANN model developed in this paper can be useful in the appropriate selection of the most *flexible* DLS method for achieving a desired level of robustness for a given application and execution scenario. The proposed ANN model can also be adapted to include additional input attributes (such as makespan, cost, power consumption, and others) for predicting the robustness, performance and execution cost of using a specific DLS method.

In conclusion, based on the studies performed as a part of the work described above in this chapter, the DLS robustness is a postmortem analysis of the performance of the execution of scientific applications on parallel and distributed computing system. The usefulness of the DLS robustness analysis is applicable to a specific class of time-stepping scientific applications, where the scheduling system has the ability to make decisions for selecting the most robust DLS technique at runtime based on the feedback recieved in the form of the execution performance from the previous time-step of the application. Often, such scheduling changes at runtime add a significant overhead cost to the execution performance. Therefore, a more concrete analysis of robustness is required at the initial mapping stage that can be used as a more generic way to measure robustness of scheduling methodologies. This led to the conceptualization of a more foundational work towards he study of process algebra and its use for performance modeling of static resource allocations for initial mapping of applications on parallel and heterogeneous machines.

CHAPTER 4

ROBUSTNESS ANALYSIS VIA PERFORMANCE MODELING USING A STOCHASTIC PROCESS ALGEBRA

4.1 A Study of Robustness of Resource Allocations in Parallel Computing Systems using Performance Modeling

Herein, a modeling study is presented to evaluate the robustness of resource allocations, obtained via performance modeling formalism provided by the PEPA language, in a parallel computing system. The resource allocation system considered for this study is comprised of independent parallel applications that are waiting in a job queue, parallel machines (that contain K heterogeneous processors, where K varies from machine to machine), and a set of possible resource allocation mappings. The mappings are obtained generated upon the analysis of a matrix of the estimated execution times of the applications on the machines. This is called the expected time to compute (ETC) matrix, where the entry (i, j) is the expected execution time of application i on machine j. In an ETC matrix, the row elements are the estimates of the expected execution times of a given application on different machines, and the column elements are the estimates of the expected execution times of different applications on a given machine [8]. In general, in parallel and distributed computing, it is realistic to assume that the ETC values of applications on all the available machines are known a priori. Often, these estimates are derived from applica-

tion profiling and machine benchmarking, from the previous executions of an application on a machine, or are provided by the user [52][69][74][83][86]. All the applications in the job queue have a workload associated with them and are assumed to start executing at time $t = 0$ seconds. A resource allocation mapping is used to allocate computing resources (in this case, machines) to each application in the job queue. Each application receives data at a certain rate, which is also known as the workload associated with that application. The perturbation parameter for the robustness analysis is defined as the variation in the workload of the applications.

Robustness of a resource allocation mapping is defined as the probability that the execution of the applications completes by a predefined makespan goal in the presence of perturbations. Given, A: set of parallel applications, $i \in A$: a parallel application, β_i^{max}: user defined makespan goal for i, P: set of parallel processors, $j \in P$: processors in a machine M allocated to i, $\hat{\lambda}_i$: perturbation parameter defined as workload variation from the initial workload (λ_i) for an application i, $F_i(M_j, \lambda_i)$: finishing time of application i, then the robustness (ψ) of a mapping is formulated as Equation 4.1.

$$\psi = \min_{\forall i \in A} \Pr[F_i(M_j, \lambda_i) \leq \beta_i^{max}] \tag{4.1}$$

Formally, for a given number of machines(M), a given number of applications (A), and an expected range of variations in the system workload, the robustness or a resource allocation mapping is calculated as follows:

- Define the performance feature for which the robustness needs to be measured. In this study, the performance feature is the pre-defined makespan goal, β_i^{max}.
- Define the perturbation parameter against which the performance of the system is measured for a robustness analysis of the resource allocation. In this study, the perturbation parameter is the runtime variation in the system workload ($\hat{\lambda}$), which also

translates into errors in the calculations of ETC values used for mapping applications to machines.

- Define the impact of the perturbation parameter on the performance feature. In this study, the impact of runtime variations in the system workload has an impact on the machine finishing times ($F_i(M_j, \lambda_i)$) and hence the system makespan. The ETC errors lead to an increase in the machine finishing times under the current resource allocation mapping. Therefore, the impact can be defined as the increased system makespan ($\max_{\forall i \in A, \forall j \in M}[F_i(M_j, \hat{\lambda}_i)]$). The machine finishing times and the makespan values are calculated using PEPA performance models, which are described later in this chapter and Chapter 5.
- Define the easure for the robustness analysis of the resource allocation mapping. In this study, the robustness (ψ) of a mapping is formulated as Equation 4.1, which is the smallest probability that the actual system makespan is less than or equal to the makespan goal, β_i^{max}.

The goal of the robustness analysis study is to find a resource allocation that maximizes the robustness of the execution of the applications on the assigned parallel computing machines. The robustness of a resource allocation is studied for the cases where, (i) the workload for all applications varies at the same rate, and (ii) the workload for an application varies independently of the other applications. The robustness model for a resource allocation mapping is composed as a cooperation model of the independent parallel applications and the parallel machines, which are modeled as PEPA components. Further, the parallel machines are modeled as a parallel composition of the processors in that machine. The application and the machines cooperate on all the activities that are associated with the processors in the machine allocated to that application. A high level example of a PEPA model of a mapping system for two applications (A_1, A_2) and five processors (P_0, P_1, P_2, P_3, P_4) distributed among two machines (M_1, M_2) is illustrated below. The example model is composed of application and processor components and each component has one state of $compute$ activity associated with it (one state per component). The under-

lying transition diagram for the example PEPA model is illustrated as an activity transition in Figure 4.1.

$$
\begin{aligned}
A_1 &\stackrel{def}{=} (compute_1, \top).RETURN \\
A_2 &\stackrel{def}{=} (compute_2, \top).RETURN \\
P_0 &\stackrel{def}{=} (compute_1, r_1).RETURN \\
P_1 &\stackrel{def}{=} (compute_1, r_1).RETURN \\
P_2 &\stackrel{def}{=} (compute_2, r_2).RETURN \\
P_3 &\stackrel{def}{=} (compute_2, r_2).RETURN \\
P_4 &\stackrel{def}{=} (compute_2, r_2).RETURN \\
M_1 &\stackrel{def}{=} P_0 \parallel P_1 \\
M_2 &\stackrel{def}{=} P_2 \parallel P_3 \parallel P_4 \\
Mapping &\stackrel{def}{=} (A_1 \parallel A_2) \underset{\mathcal{L}}{\bowtie} (M_1 \parallel M_2)
\end{aligned}
$$

where $\mathcal{L} = \{compute_1, compute_2\}$

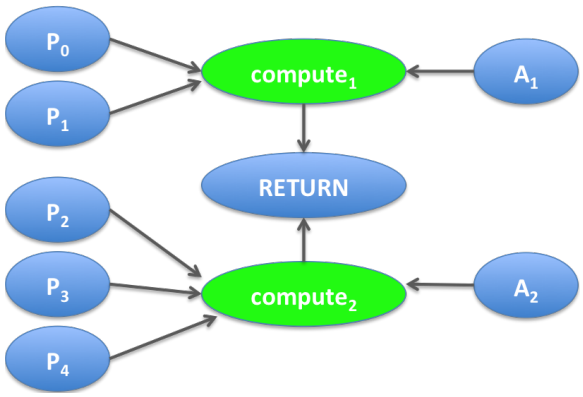

Figure 4.1: Activity diagram of an example PEPA model for a mapping system.

$\top$ is a predefined symbol in PEPA that denotes an unknown rate for an activity. The rate (r_1 or r_2) of the $compute$ activity is calculated as a function of the speed of the processors in the machine allocated to the application and the workload for that application. Therefore, for a PEPA model of a resource allocation n parallel computing systems, these rates (r_1 and r_2) are the computational rate of a machine for processing the workload of applications assigned to that machine. The computational rates also represent the compound effect of the variations in the application (workload variation) and the system (machine availabilities) characteristics. A detailed explanation of this calculation of activity rates is included in Chapter 5 along with the discussion of the experiment test cases.

The PEPA model of the resource allocation mapping becomes an input to the PEPA workbench [53]. The model components are translated into an underlying mathematical Markovian model by the PEPA workbench. The robustness of the modeled resource allocation mapping is obtained as a probability of a predefined makespan value, which is calculated by solving the Markovian model for a passage time analysis of the computational activities of all the machines. The solution is generated by the PEPA workbench as a cumulative distribution function (CDF) of the machine finishing times for the modeled resource allocation mapping. Further, the robustness of the mapping is obtained, as the minimum probability of achieving a user defined makespan goal, from the generated CDFs.

4.2 PEPA Performance Models of Resource Allocations in Parallel Computing Systems

As aforementioned, the robustness analysis of resource allocations is categorized into two test cases: (i) when the workload for all applications varies at the same rate, and (ii) when the workload for an application varies independently of the other applications. In the first test case, PEPA models are developed to mimic the functionality of the experiments related to the state-of-the-art study of robustness of static resource allocations in heterogeneous computing systems [3]. A comparison of the results of the numerical evaluation of the PEPA model with the results obtained using state-of-the-art experiments that validates the model design and the associated robustness analysis is discussed in detail in Chapter 5. In the second test case, for the sake of a comparative analysis, PEPA models are developed for the same mappings that are used in the first test case. However, the perturbation parameter, which s an application workload, varies non-uniformly and independently from the workloads of other applications in the same job queue. A PEPA model has been developed for every feasible mapping of the 20 applications to the 5 heterogeneous machines. Each PEPA model captures an execution scenario for a probable workload variation for a given mapping. Following is a detailed description of the PEPA models developed for analyzing the robustness of two mappings for allocating 5 heterogeneous machines to 20 independent applications. The two mappings, denoted as `Mapping A` and `Mapping B`, are modeled using PEPA for the two test case scenarios. `Mapping A` and `Mapping B` are used for validation of the PEPA models and are analogous to the mappings used for the robustness analysis via direct experimentation in [3].

4.2.1 PEPA modeling for case study (i): equal workload variation across all applications

The case study is designed to investigate the robustness, using the performance modeling provided by PEPA, of resource allocations (mappings of applications to computational machines) in a class of parallel heterogeneous computing systems that are prone to perturbations in the form of unpredictable variations in the system workload. For this case study, the variations in the workload are considered to be equal across all applications. The parallel computing system consists of 20 independent applications, 5 heterogeneous machines, and a set of 3 heterogeneous sensors that produce the workload ($\lambda_1, \lambda_2, and \lambda_3$) for a data set that is executed by an application. Several mappings are generated based on the ETC values of an application on a given machine. The ETC values are known a priori and are calculated as a function of the compound effect of the application workload and the computational availability of the machines. The computational availabilities are obtained through a profiling of the historical data of the performance of that machine. For a given data set, the ETC value of an application i on machine j is calculated assuming that i is the only application executing on j. For comparison and validation purposes, the ETC values, used for generating the PEPA model of mappings in this case study, and the values of the initial and varying application workload are obtained from the experimental data used in [3]. These ETC values were generated by sampling a gamma distribution, where the characteristic shape parameter, α, and scale parameter, β, were derived using three parameters: mean ($\mu = 10$), machine heterogeneity $= 0.7$, task heterogeneity $= 0.7$ [8][3]. The two mappings used for validation of the PEPA models are given in Table 4.1.

Table 4.1: Mapping A and Mapping B of applications (a_i) to machines (m_j) based on the initial sensor load values: $\lambda_1 = 962, \lambda_2 = 380, and \lambda_3 = 240$.

Machine	Mapping A	Mapping B
m_1	a_5, a_9, a_{12}, a_{17}, a_{20}	a_3, a_4, a_5, a_{17}, a_{18}, a_{20}
m_2	a_6, a_{16}	a_2, a_{11}, a_{14}, a_{19}
m_3	a_1, a_3, a_7	a_1, a_7, a_{13}
m_4	a_2, a_4, a_{10}, a_{13}, a_{15}, a_{19}	a_9, a_{12}, a_{15}
m_5	a_8, a_{11}, a_{14}, a_{18}	a_6, a_8, a_{10}, a_{16}

In this case study, both `Mapping A` and `Mapping B` are modeled individually using PEPA. Therefore, two separate PEPA models are created to numerically produce the functionality of the execution of applications on the assigned machines based on `Mapping A` and `Mapping B`. The two PEPA models are illustrated in Figure 4.2 and Figure 4.3.

As illustrated in the two PEPA models, the applications ($A1 \cdots A20$) are defined as PEPA components that engage in the *compute* activity. The rates of all the *compute* activities across all the applications are unspecified, as denoted by the PEPA symbol $\top$, since the computation times depend on the machines that are executing the corresponding applications. Further, the machines ($M1 \cdots M5$) are defined as components that engage in multiple *compute* activities that are associated with the applications assigned to the machine. Each machine is modeled using the PEPA *choice* ($+$) operator, which represents the two possible execution scenarios for a machine. The left hand side of the $+$ operator models the ideal execution scenario in the absence of perturbations in the initial application workload, where $\hat{\lambda_i} = \lambda_i, \forall i \in \{1, 2, 3\}$. The right hand side of the $+$ operator models the perturbed execution scenario in the presence of equal variations in the workload across all applications, where $\hat{\lambda_i} \neq \lambda_i$. The rates of the *compute* activities are calculated as a

$$
\begin{aligned}
A1 &\stackrel{def}{=} (compute1, \top).A1 \\
A2 &\stackrel{def}{=} (compute2, \top).A2 \\
A3 &\stackrel{def}{=} (compute3, \top).A3 \\
A4 &\stackrel{def}{=} (compute4, \top).A4 \\
A5 &\stackrel{def}{=} (compute5, \top).A5 \\
A6 &\stackrel{def}{=} (compute6, \top).A6 \\
A7 &\stackrel{def}{=} (compute7, \top).A7 \\
A8 &\stackrel{def}{=} (compute8, \top).A8 \\
A9 &\stackrel{def}{=} (compute9, \top).A9 \\
A10 &\stackrel{def}{=} (compute10, \top).A10 \\
A11 &\stackrel{def}{=} (compute11, \top).A11 \\
A12 &\stackrel{def}{=} (compute12, \top).A12 \\
A13 &\stackrel{def}{=} (compute13, \top).A13 \\
A14 &\stackrel{def}{=} (compute14, \top).A14 \\
A15 &\stackrel{def}{=} (compute15, \top).A15 \\
A16 &\stackrel{def}{=} (compute16, \top).A16 \\
A17 &\stackrel{def}{=} (compute17, \top).A17 \\
A18 &\stackrel{def}{=} (compute18, \top).A18 \\
A19 &\stackrel{def}{=} (compute19, \top).A19 \\
A20 &\stackrel{def}{=} (compute20, \top).A20 \\
M1 &\stackrel{def}{=} (compute5, r5).(compute9, r9).(compute12, r12). \\
&\quad (compute17, r17).(compute20, r20).M1 \\
&+ (compute5, p5).(compute9, p9).(compute12, p12). \\
&\quad (compute17, p17).(compute20, p20).M1 \\
M2 &\stackrel{def}{=} (compute6, r6).(compute16, r16).M2 \\
&+ (compute6, p6).(compute16, p16).M2 \\
M3 &\stackrel{def}{=} (compute1, r1).(compute3, r3).(compute7, r7).M3 \\
&+ (compute1, p1).(compute3, p3).(compute7, p7).M3 \\
M4 &\stackrel{def}{=} (compute2, r2).(compute4, r4).(compute10, r10). \\
&\quad (compute13, r13).(compute15, r15).(compute19, r19).M4 \\
&+ (compute2, p2).(compute4, p4).(compute10, p10). \\
&\quad (compute13, p13).(compute15, p15).(compute19, p19).M4 \\
M5 &\stackrel{def}{=} (compute8, r8).(compute11, r11).(compute14, r14).(compute18, r18).M5 \\
&+ (compute8, p8).(compute11, p11).(compute14, p14).(compute18, p18).M5 \\
Mapping &\stackrel{def}{=} A1 \parallel A2 \parallel A3 \parallel A4 \parallel A5 \parallel A6 \parallel A7 \parallel A8 \parallel A9 \parallel A10 \parallel A11 \\
&\quad \parallel A12 \parallel A13 \parallel A14 \parallel A15 \parallel A16 \parallel A17 \parallel A18 \parallel A19 \parallel A20 \\
&\quad \bowtie_{\mathcal{L}} M1 \parallel M2 \parallel M3 \parallel M4 \parallel M5
\end{aligned}
$$

where, $\mathcal{L} = \{compute1, \cdots, compute20\}$

Figure 4.2: PEPA model for Mapping A defined as a cooperation between the applications and the machines over the *compute* activity.

function of λ_i and the actual computation times ($T_i, \forall i \in \{1 \cdots 20\}$) of each application on the machine where it is mapped. Further, T_i is calculated as a function of the machine availabilities (generated by sampling a gamma distribution) and $\hat{\lambda}_i$. In the ideal execution

$$
\begin{aligned}
A1 &\stackrel{def}{=} (compute1, \top).A1 \\
A2 &\stackrel{def}{=} (compute2, \top).A2 \\
A3 &\stackrel{def}{=} (compute3, \top).A3 \\
A4 &\stackrel{def}{=} (compute4, \top).A4 \\
A5 &\stackrel{def}{=} (compute5, \top).A5 \\
A6 &\stackrel{def}{=} (compute6, \top).A6 \\
A7 &\stackrel{def}{=} (compute7, \top).A7 \\
A8 &\stackrel{def}{=} (compute8, \top).A8 \\
A9 &\stackrel{def}{=} (compute9, \top).A9 \\
A10 &\stackrel{def}{=} (compute10, \top).A10 \\
A11 &\stackrel{def}{=} (compute11, \top).A11 \\
A12 &\stackrel{def}{=} (compute12, \top).A12 \\
A13 &\stackrel{def}{=} (compute13, \top).A13 \\
A14 &\stackrel{def}{=} (compute14, \top).A14 \\
A15 &\stackrel{def}{=} (compute15, \top).A15 \\
A16 &\stackrel{def}{=} (compute16, \top).A16 \\
A17 &\stackrel{def}{=} (compute17, \top).A17 \\
A18 &\stackrel{def}{=} (compute18, \top).A18 \\
A19 &\stackrel{def}{=} (compute19, \top).A19 \\
A20 &\stackrel{def}{=} (compute20, \top).A20 \\
M1 &\stackrel{def}{=} (compute3, r3).(compute4, r4).(compute5, r5). \\
&\quad (compute17, r17).(compute18, r18).(compute20, r20).M1 \\
&+ (compute3, p3).(compute4, p4).(compute5, p5). \\
&\quad (compute17, p17).(compute18, p18).(compute20, p20).M1 \\
M2 &\stackrel{def}{=} (compute2, r2).(compute11, r11).(compute14, r14).(compute19, r19).M2 \\
&+ (compute2, p2).(compute11, p11).(compute14, p14).(compute19, p19).M2 \\
M3 &\stackrel{def}{=} (compute1, r1).(compute7, r7).(compute13, r13).M3 \\
&+ (compute1, p1).(compute7, p7).(compute13, p13).M3 \\
M4 &\stackrel{def}{=} (compute9, r9).(compute12, r12).(compute15, r15).M4 \\
&+ (compute9, p9).(compute12, p12).(compute15, p15).M4 \\
M5 &\stackrel{def}{=} (compute6, r6).(compute8, r8).(compute10, r10).(compute16, r16).M5 \\
&+ (compute6, p6).(compute8, p8).(compute10, p10).(compute16, p16).M5 \\
Mapping &\stackrel{def}{=} A1 \parallel A2 \parallel A3 \parallel A4 \parallel A5 \parallel A6 \parallel A7 \parallel A8 \parallel A9 \parallel A10 \parallel A11 \\
&\quad \parallel A12 \parallel A13 \parallel A14 \parallel A15 \parallel A16 \parallel A17 \parallel A18 \parallel A19 \parallel A20 \\
&\quad \bowtie_{\mathcal{L}} M1 \parallel M2 \parallel M3 \parallel M4 \parallel M5
\end{aligned}
$$

where, $\mathcal{L} = \{compute1, \cdots, compute20\}$

Figure 4.3: PEPA model for Mapping B defined as a cooperation between the applications and the machines over the *compute* activity.

scenario, $\hat{\lambda}_i = \lambda_i, \forall i \in \{1, 2, 3\}$. Therefore, T_i is equal to the initial ETC values and consequently, the rates, $r1 \cdots r20$, are only calculated using the machine availability values. However, in the perturbed execution scenario (where, $\hat{\lambda}_i \neq \lambda_i$) the rates $p1 \cdots p20$ are calculated differently than the rates $r1 \cdots r20$. Due to the variation in the workload, T_i values

vary from the ETC values of the applications on the assigned machines. Therefore, the perturbed rates, $p1 \cdots p20$, are calculated using the machine availabilities, λ_i, and $\hat{\lambda}_i$ values. A more detailed description of the calculation of the rates of the *compute* activities is given in Chapter 5. The overall mapping is defined in the last statement of the PEPA model given in Figures 4.2 and 4.3. The mapping itself is modeled as a separate PEPA component, which results from a cooperation of the application and the machine components over the cooperation set of all the associated *compute* activities. In addition, the applications and the machines are modeled as independent parallel components in the cooperation function.

4.2.2 PEPA modeling for case study (ii): non-uniform workload variation across all applications

Similar to the previous case study, this case study is also designed to investigate the robustness, using the performance modeling provided by PEPA, of resource allocations (mappings of applications to computational machines) in a class of parallel heterogeneous computing systems that are prone to perturbations in the form of unpredictable variations in the system workload. However, in this case study, the workload vary non-uniformly across different applications. PEPA models are created for for 20 independent applications, 5 heterogeneous machines, and a set of 3 heterogeneous sensors that produce the workload ($\lambda_1, \lambda_2, and \lambda_3$). For comparison purpose, PEPA models are generated for the same mappings that were modeled in the previous case study as shown in Table 4.1. The semantics of the new PEPA models for these mappings remain similar to the ones as illustrated in Figure 4.2 and 4.3. However, the values of the perturbed rates, $p1 \cdots p20$, are calculated using different values of $\hat{\lambda}_i$. Different random values, for $\hat{\lambda}_i$, are sampled from a

mixture distribution (gamma, Gaussian, Erlang-K, and exponential) for every application for modeling a highly perturbed execution environment with a non-uniform variation of system workload. The machine availabilities are assumed to remain constant throughout the execution.

CHAPTER 5

MODELING STUDY AND ROBUSTNESS ANALYSIS

A modeling study, of resource allocations in parallel computing systems, using PEPA for performance evaluation is presented in this chapter. Performance results drawn from the modeling study are used for analyzing the robustness of candidate mappings of a number of applications to available parallel heterogeneous machines. Moreover, a comparison of the mappings, in terms of performance constraints (such as, achieving a makespan goal) and their robustness with respect to variations in workload, is also studied. The resource allocation system under consideration for this study is similar to the HiPer-D like system [56][51], which was also used in the robustness analysis performed in the state-of-the-art [3] used for validation of the modeling study. The resource allocation system consists of a job queue that contains 20 independent applications (i), at any given time, waiting to be scheduled onto one of the 5 parallel machines (j). The machines are dedicated to the applications assigned to them and together, they form a heterogeneous computing environment. The system also consists of 3 heterogeneous sensors producing load distributions ($\lambda_1, \lambda_2, \lambda_3$), which contribute to the overall system workload (λ_i). Under the assumption that the ETC values are known a priori, a number of candidate mappings are obtained for allocating applications to the available machines. A resource allocation mapping is de-

rived using the known ETC values and the initial application workload (λ_i). According to the robustness evaluation methodology as described in Chapter 4, performance in terms of the system makespan needs to be calculated for a mapping in the given resource allocation system. The calculation of the system performance is done via performance modeling and a numerical performance evaluation of the models using the stochastic process algebra, PEPA [59]. The resource allocation mapping under consideration is modeled using the high level formalism provided by PEPA, which includes a definition the applications and their characteristics, definition of the machines and their computational characteristics with respect to the actual system workload ($\hat{\lambda}_i$), and definition of the overall system as the mapping of applications to the respective machines. Further, this PEPA formalism is input to the PEPA workbench [53], where the model is compiled and translated into an underlying mathematical system of continuous time Markov chain (CTMC) processes. The execution of the mapping is modeled via these CTMC processes. The performance feature, calculated by solving the mathematical system of CTMC processes for a passage time analysis, is obtained as cumulative distribution functions (CDFs) of the finishing times of the computing machines ($F_i(M_j, \hat{\lambda}_i)$), which translate into the system makespan ($\max_{\forall i \in A, \forall j \in M}[F_i(M_j, \hat{\lambda}_i)]$). The *robustness* of the modeled mapping is then analyzed as the probability for which the calculated system makespan is less than or equal to the makespan goal (β_i^{max}).

The modeling study is divided into two case studies: (i) validation case study, is used to validate the robustness analysis and the numerical performance modeling and evaluation against the results of the robustness analysis performed in experiments presented in the

considered state-of-the-art [3], (ii) robustness evaluation case study, is used to evaluate the robustness of the mappings modeled in the previous case under a highly perturbed execution environment, where the sensors produce data at rapidly varying rates ($\lambda_1, \lambda_2, \lambda_3$) that may lead to a highly uncertain execution environment with entirely different sensor loads for every application application. The mappings used for the two case studies are given in Table 4.1. The modeling case studies are discussed in more detail in the following sections.

5.1 Robustness evaluation case study: equal workload variation across all applications

In this study, the execution of two mappings, the two mappings as given in Table 4.1, are modeled as CTMC processes using PEPA. The PEPA input file that defines the model for the two mappings is shown in Figure 4.2 and 4.3, respectively. Further, to complete the PEPA model, the rates $r1 \cdots r20$ and $p1 \cdots p20$, need to be calculated for solving the underlying Markovian model via a passage time analysis using the PEPA workbench to derive performance measures.

5.1.1 Deriving PEPA activity rates in ideal computing environment ($\hat{\lambda} = \lambda$)

The rates associated with the $compute_i$ activities, when there is no variation in the application workload, are represented by $r1 \cdots r20$. The calculation for the rate is given in Equation 5.1, where T_{ij} is the actual time to compute an application i on machine j, $\hat{\lambda_i}$ is calculated as a function of the varying sensor loads $\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$, and λ_i is calculated as a function of the initial sensor loads $\lambda_1, \lambda_2, \lambda_3$.

$$r_i = \frac{\lambda_i}{T_{ij}} \quad \forall i, j, \; where \; T_{ij} = f(\hat{\lambda}_i = \lambda_i) \tag{5.1}$$

The T_{ij} values of the 20 applications on the machines they are assigned according to the two mappings are listed in Table 5.1 and 5.2, respectively [3]. T_{ij} is calculated as a product of the machine availability factor (η_i), which is the computational availability of the allocated machine j for executing application i, and the runtime workload for that application ($\hat{\lambda}_i$). The right most column of the table also lists the corresponding value of the ideal rate r_i at which the allocated machine executes application a_i. r_i is calculated as a ratio of the initial workload λ_i and T_{ij}. This signifies that the rate at which a machine will compute an assigned application is directly proportional to the workload value according to which the application was initially allocated to that machine and concurrently, is inversely affected by the actual computational time, which is dependent upon the actual (initial or varied) sensor loads during the execution of that application. In the ideal computing scenario, where $\hat{\lambda}_i = \lambda_i$, the value of r_i reduces to an inverse of the machine availability factor (η_i). The initial sensor load values are, $\lambda_1 = 962$, $\lambda_2 = 380$, and $\lambda_3 = 240$.

5.1.2 Deriving PEPA activity rates in perturbed computing environment ($\hat{\lambda} \neq \lambda$)

The rates associated with the $compute_i$ activities, in the presence of perturbations in the form of varying application workload, are represented by $p1 \cdots p20$. The calculation for the rate is given in Equation 5.2, where T_{ij} is the actual time to compute an application i on machine j, $\hat{\lambda}_i$ is calculated as a function of the varying sensor loads $\hat{\lambda}_1, \hat{\lambda}_2, \hat{\lambda}_3$, and λ_i is calculated as a function of the initial sensor loads $\lambda_1, \lambda_2, \lambda_3$.

Table 5.1: Example T_{ij} and r_i values, as a function of runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$) and the machine availability factor (η) for Mapping A.

Application	$T_{ij} = \eta_i(\hat{\lambda_i})$	$r_i = \frac{\lambda_i}{T_{ij}} = \frac{1}{\eta_i}$
a_1	$3.90(4\hat{\lambda_3})$	$1/3.90$
a_2	$7.80(5\hat{\lambda_2})$	$1/7.80$
a_3	$3.90(6\hat{\lambda_1})$	$1/3.90$
a_4	$7.80(\hat{\lambda_1})$	$1/7.80$
a_5	$6.50(3\hat{\lambda_1} + \hat{\lambda_3})$	$1/6.50$
a_6	$2.60(\hat{\lambda_3})$	$1/2.60$
a_7	$3.90(5\hat{\lambda_2})$	$1/3.90$
a_8	$5.20(6\hat{\lambda_2})$	$1/5.20$
a_9	$6.50(20\hat{\lambda_3})$	$1/6.50$
a_{10}	$7.80(5\hat{\lambda_2} + 7\hat{\lambda_3})$	$1/7.80$

Table 5.2: Example T_{ij} and r_i values, as a function of runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$) and the machine availability factor (η) for Mapping B.

Application	$T_{ij} = \eta_i(\hat{\lambda_i})$	$r_i = \frac{\lambda_i}{T_{ij}} = \frac{1}{\eta_i}$
a_1	$3.90(4\hat{\lambda_3})$	$\frac{1}{3.90}$
a_2	$5.20(2\hat{\lambda_2})$	$\frac{1}{5.20}$
a_3	$7.80(11\hat{\lambda_1})$	$\frac{1}{7.80}$
a_4	$7.80(4\hat{\lambda_1} + 2\hat{\lambda_2})$	$\frac{1}{7.80}$
a_5	$7.80(3\hat{\lambda_1} + \hat{\lambda_3})$	$\frac{1}{7.80}$
a_6	$5.20(\hat{\lambda_3})$	$\frac{1}{5.20}$
a_7	$3.90(5\hat{\lambda_2})$	$\frac{1}{3.90}$
a_8	$5.20(6\hat{\lambda_2})$	$\frac{1}{5.20}$
a_9	$3.90(3\hat{\lambda_3})$	$\frac{1}{3.90}$
a_{10}	$5.20(3\hat{\lambda_2} + 3\hat{\lambda_3})$	$\frac{1}{5.20}$

$$p_i = \frac{\lambda_i}{T_{ij}} \quad \forall i, j, \text{ where } T_{ij} = f(\hat{\lambda_i} \neq \lambda_i) \tag{5.2}$$

The T_{ij} values of the 20 applications on the machines they are assigned to according to the two mappings are listed in Table 5.3 and 5.4, respectively [3]. T_{ij} is calculated as a product of the machine availability factor (η_i), which is the computational availability of the allocated machine j for executing application i, and the runtime workload for that application ($\hat{\lambda_i}$). The runtime application workload is calculated as a function of the three runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$). The right most column of the table also lists the corresponding value of the perturbed rate p_i at which the allocated machine executes application a_i. p_i is calculated as a ratio of the initial workload λ_i and T_{ij}. This signifies that the rate at which a machine will compute an assigned application is directly proportional to the workload value according to which the application was initially allocated to that machine and concurrently, is inversely affected by the actual computational time, which is dependent upon the actual (initial or varied) sensor loads during the execution of that application. In the perturbed computing scenario, $\hat{\lambda_i} \neq \lambda_i$. The initial sensor load values are, $\lambda_1 = 962$, $\lambda_2 = 380$, and $\lambda_3 = 240$. The runtime sensor load values are set at, $\hat{\lambda_1} = 962$, $\hat{\lambda_2} = 1546$, and $\hat{\lambda_3} = 593$, for all the 20 applications to match the specifications in the experiments done in [3]. This also represents a case study for a resource allocation system that has a low load imbalance factor.

Table 5.3: Example T_{ij} and p_i values, as a function of runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$) and the machine availability factor (η) for Mapping A.

Application	$T_{ij} = \eta_i(\hat{\lambda_i})$	$r_i = \frac{\lambda_i}{T_{ij}} = \frac{\lambda_i}{\eta_i(\hat{\lambda_i})}$
a_1	$3.90(4\hat{\lambda_3})$	$\frac{4\lambda_3}{3.90(4\hat{\lambda_3})}$
a_2	$7.80(5\hat{\lambda_2})$	$\frac{5\lambda_2}{7.80(5\hat{\lambda_2})}$
a_3	$3.90(6\hat{\lambda_1})$	$\frac{6\lambda_1}{3.90(6\hat{\lambda_1})}$
a_4	$7.80(\hat{\lambda_1})$	$\frac{\lambda_1}{7.80(\hat{\lambda_1})}$
a_5	$6.50(3\hat{\lambda_1} + \hat{\lambda_3})$	$\frac{3\lambda_1+\lambda_3}{6.50(3\hat{\lambda_1}+\hat{\lambda_3})}$
a_6	$2.60(\hat{\lambda_3})$	$\frac{\lambda_3}{2.60(\hat{\lambda_3})}$
a_7	$3.90(5\hat{\lambda_2})$	$\frac{5\lambda_2}{3.90(5\hat{\lambda_2})}$
a_8	$5.20(6\hat{\lambda_2})$	$\frac{6\lambda_2}{5.20(6\hat{\lambda_2})}$
a_9	$6.50(20\hat{\lambda_3})$	$\frac{20\lambda_3}{6.50(20\hat{\lambda_3})}$
a_{10}	$7.80(5\hat{\lambda_2} + 7\hat{\lambda_3})$	$\frac{5\lambda_2+7\lambda_3}{7.80(5\hat{\lambda_2}+7\hat{\lambda_3})}$

Table 5.4: Example T_{ij} and p_i values, as a function of runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$) and the machine availability factor (η) for Mapping B.

Application	$T_{ij} = \eta_i(\hat{\lambda_i})$	$r_i = \frac{\lambda_i}{T_{ij}} = \frac{\lambda_i}{\eta_i(\hat{\lambda_i})}$
a_1	$3.90(4\hat{\lambda_3})$	$\frac{4\lambda_3}{3.90(4\hat{\lambda_3})}$
a_2	$5.20(2\hat{\lambda_2})$	$\frac{2\lambda_2}{5.20(2\hat{\lambda_2})}$
a_3	$7.80(11\hat{\lambda_1})$	$\frac{11\lambda_1}{7.80(11\hat{\lambda_1})}$
a_4	$7.80(4\hat{\lambda_1} + 2\hat{\lambda_2})$	$\frac{4\lambda_1+2\lambda_2}{7.80(4\hat{\lambda_1}+2\hat{\lambda_2})}$
a_5	$7.80(3\hat{\lambda_1} + \hat{\lambda_3})$	$\frac{3\lambda_1+\lambda_3}{7.80(3\hat{\lambda_1}+\hat{\lambda_3})}$
a_6	$5.20(\hat{\lambda_3})$	$\frac{\lambda_3}{5.20(\hat{\lambda_3})}$
a_7	$3.90(5\hat{\lambda_2})$	$\frac{5\lambda_2}{3.90(5\hat{\lambda_2})}$
a_8	$5.20(6\hat{\lambda_2})$	$\frac{6\lambda_2}{5.20(6\hat{\lambda_2})}$
a_9	$3.90(3\hat{\lambda_3})$	$\frac{3\lambda_3}{3.90(3\hat{\lambda_3})}$
a_{10}	$5.20(3\hat{\lambda_2} + 3\hat{\lambda_3})$	$\frac{3\lambda_2+3\lambda_3}{5.20(3\hat{\lambda_2}+3\hat{\lambda_3})}$

5.1.3 Numerical Analysis and Validation of Performance Modeling of Resource Allocations using the PEPA Workbench

Once the ideal and the perturbed rates are calculated for completing the PEPA input files representing the performance models of the execution of applications on the allocated machines with respect to the two mappings, the PEPA models are compiled and solved using the PEPA workbench tool. The PEPA input files as shown in Figure 4.2 and 4.3, along with the rates r_i and p_i calculated using Tables 5.3, 5.4, 5.1, and 5.2, are compiled using the Eclipse Luna development tool [40] in a PEPA workbench framework, as shown in Figure 5.1. After the model compiles successfully according to the PEPA formalisms, the state space of the underlying mathematical Markovian model of the execution of the mappings is derived (shown in the screenshot in Figure 5.2).

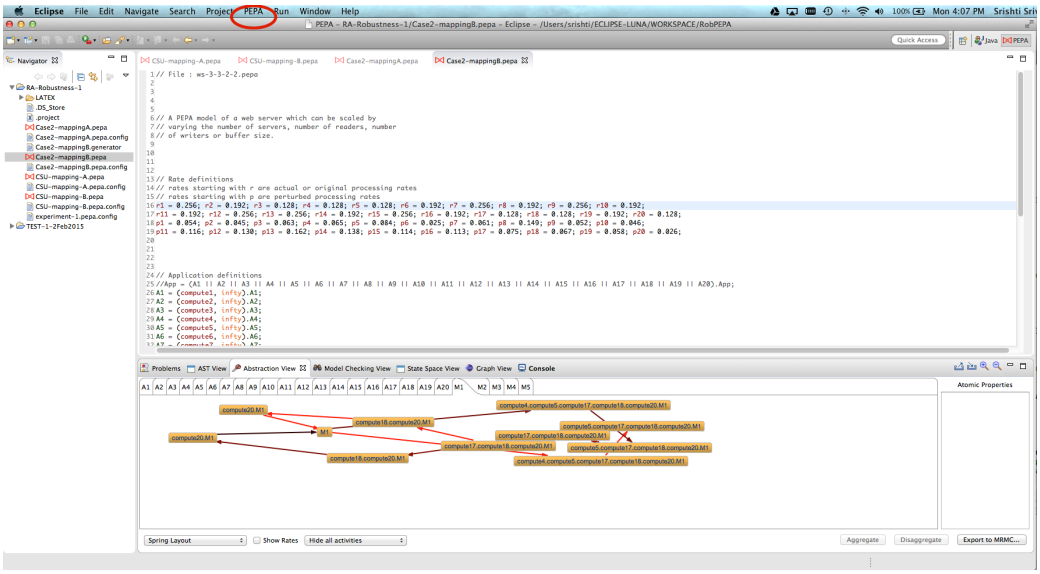

Figure 5.1: Screenshot of the Eclipse Luna Development Tool with the PEPA workbench modeling framework.

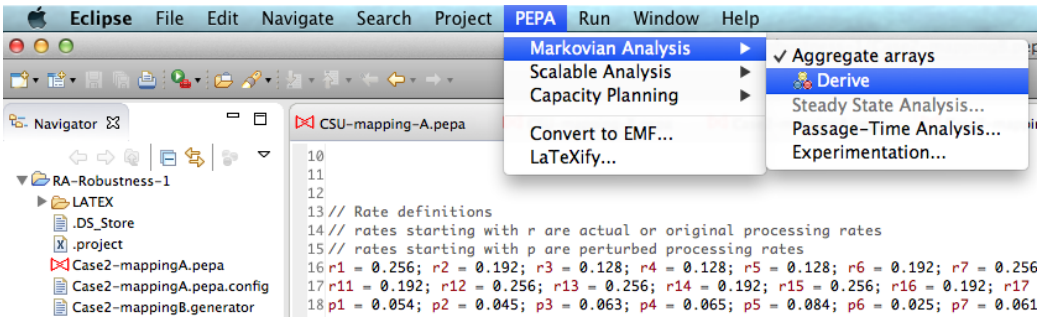

Figure 5.2: Screenshot of the derivation of the state space of the underlying mathematical Markovian model.

The state space generated for the Markov model representing the execution of the two mappings consists of 8640 and 13475 number of states, respectively, of the CTMC processes representing the execution states of the applications and the allocated machines. Examples of activity diagrams resulting from the generated state space are illustrated in Figures 5.3 and 5.4. In the activity diagram, the rectangular boxes represent the different states of a PEPA component (or the underlying CTMC process). The arcs represent the transitions between the states. The multiplicity of outgoing arcs (represented by two different colors) represent the possibility that a component may transition to either of the states at the end of one of the multiple outgoing arcs in its current state. For example, in Figure 5.3, the PEPA component M_3 has two outgoing arcs in its initial state. The two arcs represent the uncertainty that signifies the computation the application A_1 on the allocated machine M_3 in, either the *ideal* computing scenario with the rate $r_1 = 0.256$ or the *perturbed* computing scenario with the rate $p_1 = 0.104$. Every PEPA component is translated into its own activity diagram, modeling its evolution throughout the execution of the computational activities, similar to the activity diagrams shown in Figures 5.3 and 5.4.

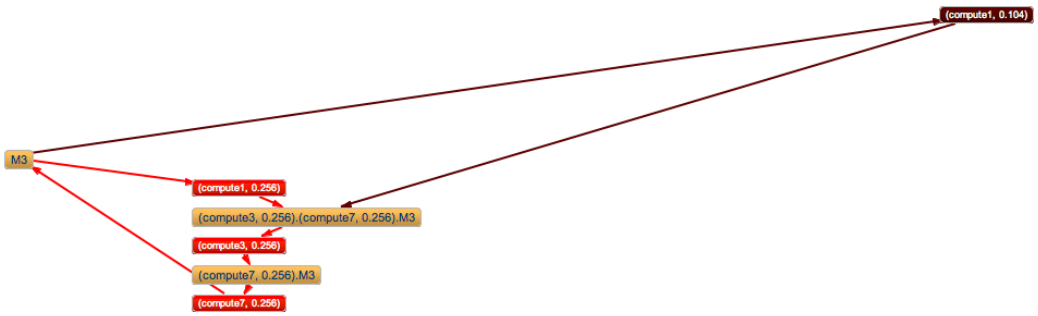

Figure 5.3: An activity diagram of the CTMC processes of the corresponding PEPA components in *Mapping A*.

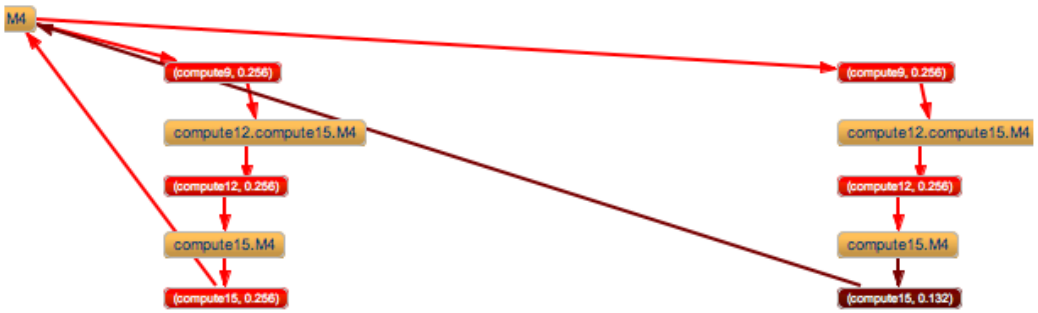

Figure 5.4: An activity diagram of the CTMC processes of the corresponding PEPA components in *Mapping B*.

Once the state space and the resulting activity diagrams of the components (CTMC processes) of the PEPA model are generated, the tool allows the modeler to specify the type of Markovian analysis that needs to be used for solving the generated Markov models to derive performance measures. As defined in Chapter 2, the PEPA workbench allows two types of Markovian analysis: (i) steady state analysis is used to solve the global balance equation (representing the state of equilibrium of the Markov model) using the infinitesimal generator matrix to calculate steady state probability values for every component to derive performance measures, such as, throughput and utilization, (ii) passage time analysis is

used to solve the Markov models using the timing information associated with the activity rates to derive performance measures, such, as makespan and response time. The passage time analysis generates a CDF of the passage time (T_p) from a source state (S_s) into a non-empty set of target states (S_T), s.t.,

$$T_p = \inf\{u > 0 : S_s(u) \in S_T | S_s(0) = \text{initial state}\} \tag{5.3}$$

The CDF is generated by convolving state holding times over all possible paths from state $i \in S_s$ into any of the states in the set S_T. In this case study, the performance feature of interest is *makespan*, therefore, passage time analysis is used to solve the underlying Markov models. Based on the knowledge that the modeler provides to the PEPA workbench in the form of PEPA formalism of the resource allocation system, the tool extracts the information from that knowledge in the form of initial state values, final state values, and the intermediate transitions. Using this information, the tool enables a passage time analysis of the Markov models for deriving the *makespan* as the passage time between the defined initial and final states. For example, as shown in Figure 5.5, the tool collects information from the PEPA model regarding the initial state(s) to be the state(s) associated with the $compute_1$ activity values and the final state(s) to be the state(s) associated with the $compute_8$ activity. The *start time* and the *stop time* values are specified the modeler and are often specified by the application user. For example, the *stop time* is analogous to the user specified makespan goal, β_i^{max}. In our modeling study, the passage time analysis of the Markov models underlying the PEPA definitions of the two resource allocation mappings, yield CDFs of the machine finishing times, $F_i(M_j, \hat{\lambda}_i)$, as passage times between the states

associated with the applications assigned to that machine. The CDFs of the finishing times of machines $M_1 \cdots M_5$ in the Markov model for `Mapping A` are shown in Figures 5.6 through 5.10, and for `Mapping B` are shown in Figures 5.11 through 5.15.

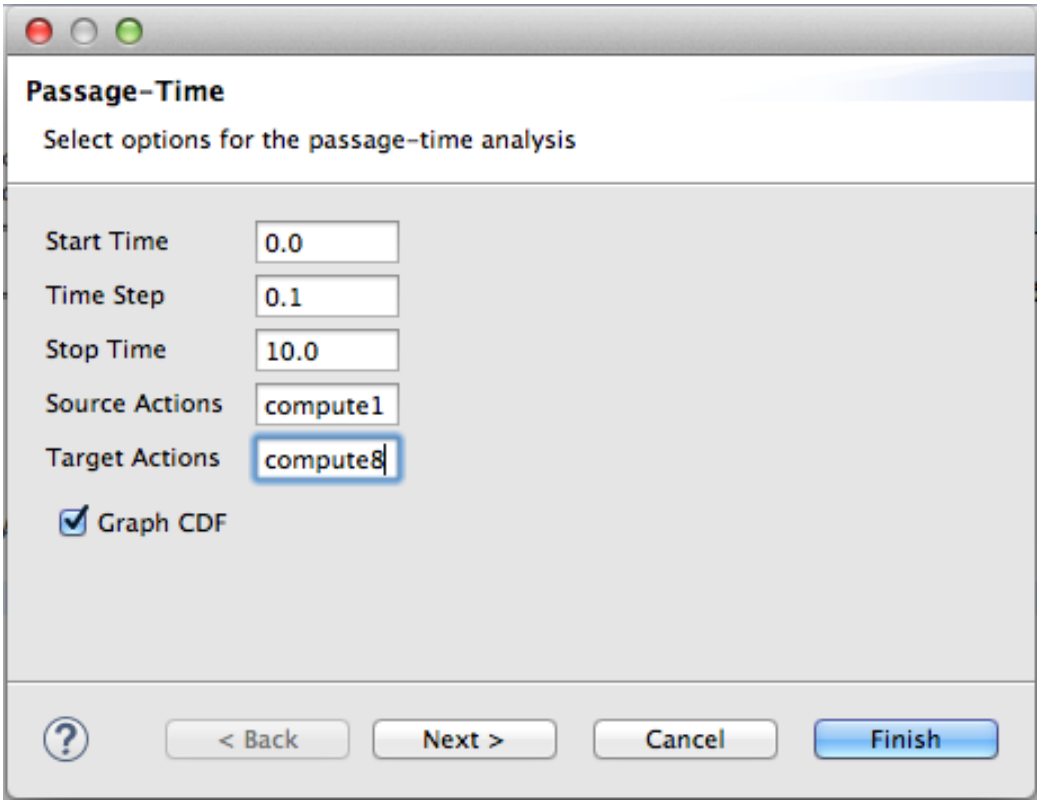

Figure 5.5: Passage time analysis parameters generated by the PEPA workbench.

For validation of performance modeling and robustness analysis, the performance modeling of resource allocations considered in the modeling study in this research has been validated via a comparative analysis of the performance results obtained using PEPA modeling and by solving the underlying numerical Markov models, with the results obtained from the experiments given in the considered state-of-the-art [3][6][5]. The metric used

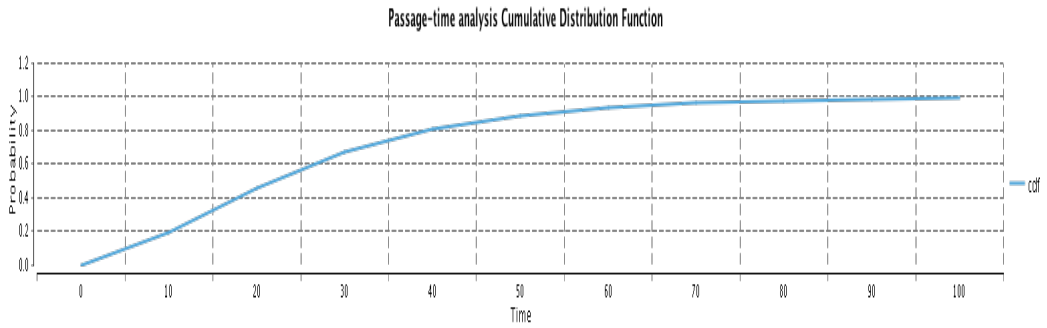

Figure 5.6: Cumulative distribution function (CDF) of the finishing time of machine M_1 for executing applications $A_5, A_9, A_{12}, A_{17}, A_{20}$ as given by *Mapping A*.

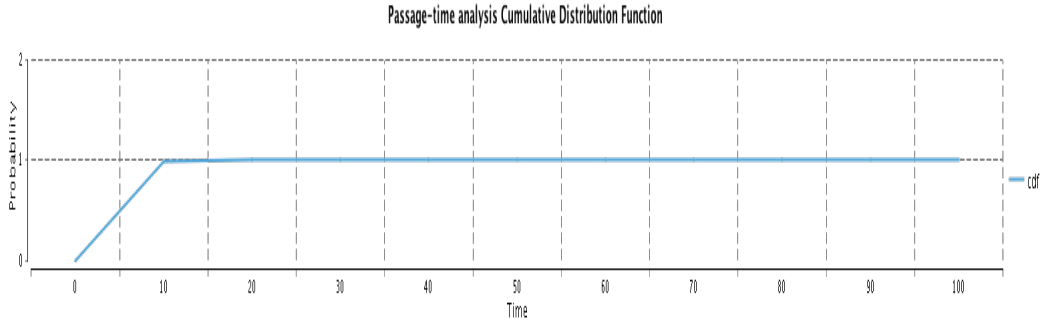

Figure 5.7: Cumulative distribution function (CDF) of the finishing time of machine M_2 for executing applications A_6, A_{16} as given by *Mapping A*.

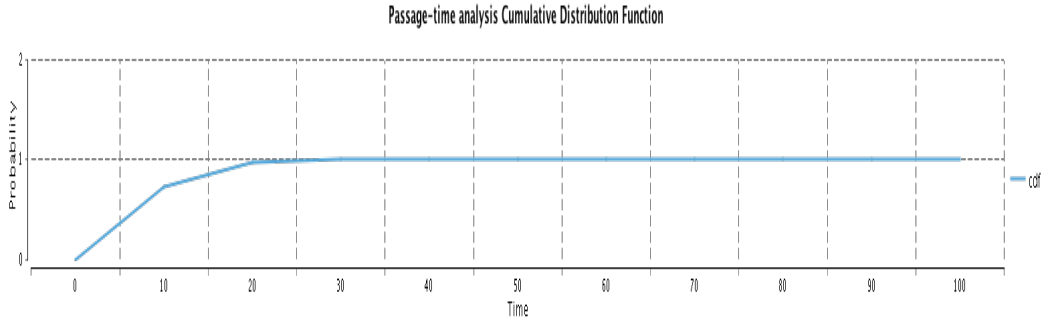

Figure 5.8: Cumulative distribution function (CDF) of the finishing time of machine M_3 for executing applications A_1, A_3, A_7 as given by *Mapping A*.

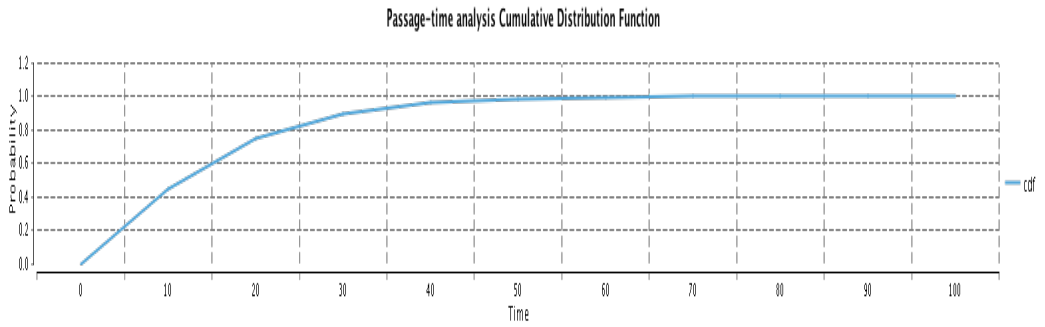

Figure 5.9: Cumulative distribution function (CDF) of the finishing time of machine M_4 for executing applications $A_2, A_4, A_{10}, A_{13}, A_{15}, A_{19}$ as given by *Mapping A*.

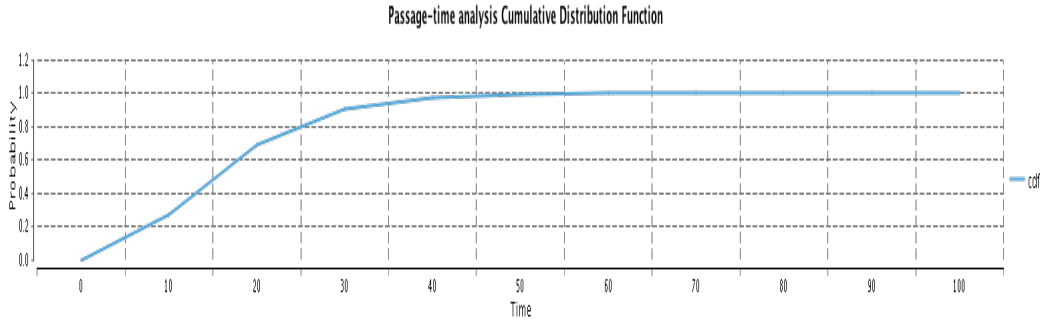

Figure 5.10: Cumulative distribution function (CDF) of the finishing time of machine M_5 for executing applications $A_8, A_{11}, A_{14}, A_{18}$ as given by *Mapping A*.

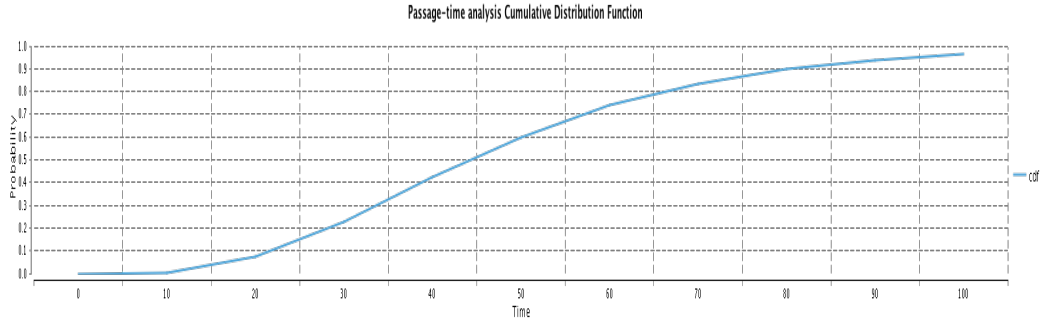

Figure 5.11: Cumulative distribution function (CDF) of the finishing time of machine M_1 for executing applications $A_3, A_4, A_5, A_{17}, A_{18}, A_{20}$ as given by *Mapping B*.

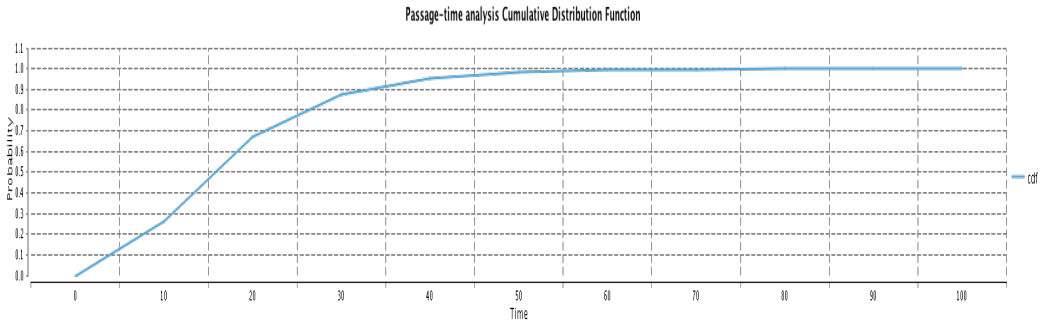

Figure 5.12: Cumulative distribution function (CDF) of the finishing time of machine M_2 for executing applications $A_2, A_{11}, A_{14}, A_{19}$ as given by *Mapping B*.

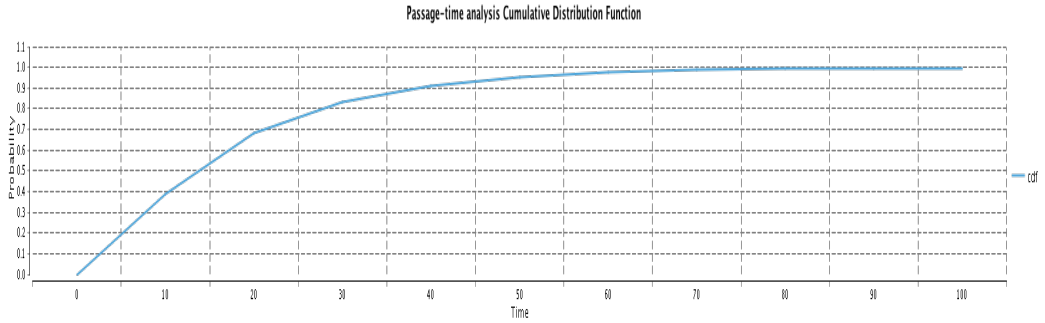

Figure 5.13: Cumulative distribution function (CDF) of the finishing time of machine M_3 for executing applications A_1, A_7, A_{13} as given by *Mapping B*.

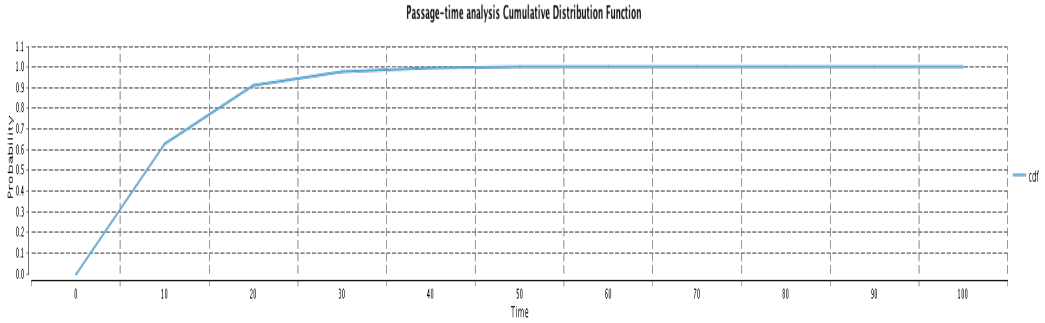

Figure 5.14: Cumulative distribution function (CDF) of the finishing time of machine M_4 for executing applications A_9, A_{12}, A_{15} as given by *Mapping B*.

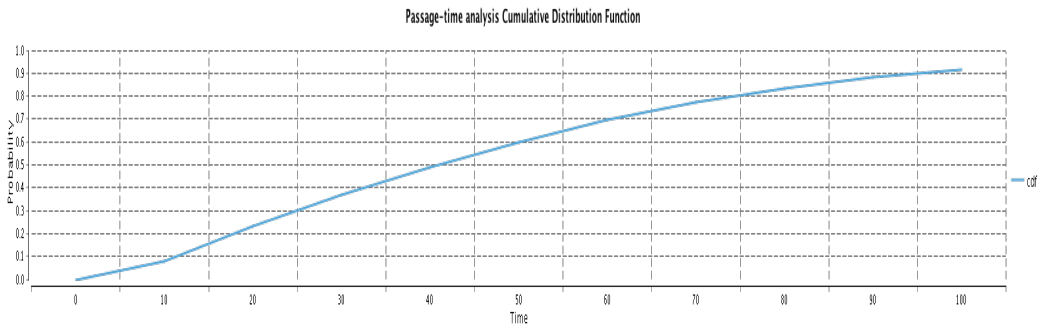

Figure 5.15: Cumulative distribution function (CDF) of the finishing time of machine M_5 for executing applications A_6, A_8, A_{10}, A_{16} as given by *Mapping B*.

for comparing the performance results are the machine finishing times. A close similarity in the performance results of the two methods (simulation experiments and analytical modeling using PEPA) of performance evaluation illustrated as a result of the comparative analysis as shown in Figure 5.16. The slight differences in the finishing times of the machines can be explained as a result of the systemic variations, from external factors such as, I/O tasks, in the actual computing machines that were used to carry out the simulation experiments. This also validates the advantage of numerical analysis over simulation experiments for obtaining more precise results.

The robustness of the resource allocation defined by the performance modeling of the two mappings is formulated using Equation 4.1. Let ψ_A and ψ_B be the robustness values of the two mappings respectively. Based on the experiments in [3], the makespan goal is set as $\beta_i^{max} = 45$ seconds. Then, the robustness of the two mappings is calculated using Equation 5.4 and 5.5. Both the two mappings yield a *makespan* of 90 seconds. However, they differ in their robustness values when the makespan goal is set to 45 seconds, where $\psi_A = 81\%$ and $\psi_B = 43\%$. The difference in the robustness values of the two

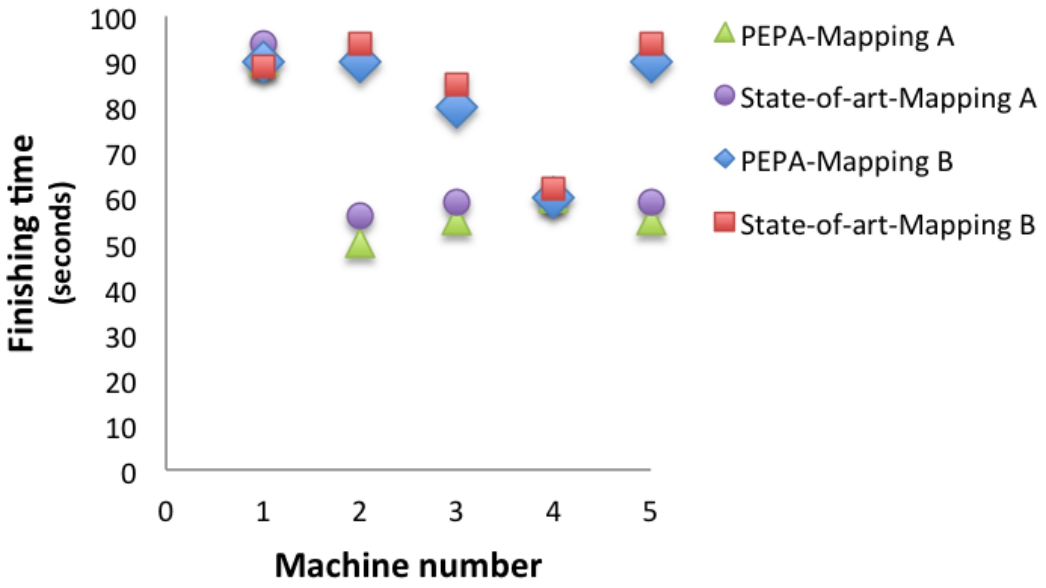

Figure 5.16: A comparative analysis of the numerical results of performance modeling with existing simulation results.

mappings that yield the same system makespan, reinstates the need for *robustness analysis* for selecting an initial mapping that can withstand the runtime perturbations in application and system parameters in a parallel computing environment. For example, in this modeling study, both the two mappings yield the same performance in the form of overall system makespan. However, `Mapping A` is a *more robust* choice for initial allocation in terms of achieving a set makespan goal in the presence of runtime perturbations, such as workload variation. An illustration of the difference in the robustness of the two mappings with the same performance value is given in Figure 5.17.

$$\psi_A = \min_{\forall i,j \; paired \; as \; in \; Mapping A} \Pr[F_i(M_j, \lambda_i) \leq 45] \tag{5.4}$$

$$\psi_B = \min_{\forall i,j \; paired \; as \; in \; Mapping B} \Pr[F_i(M_j, \lambda_i) \leq 45] \tag{5.5}$$

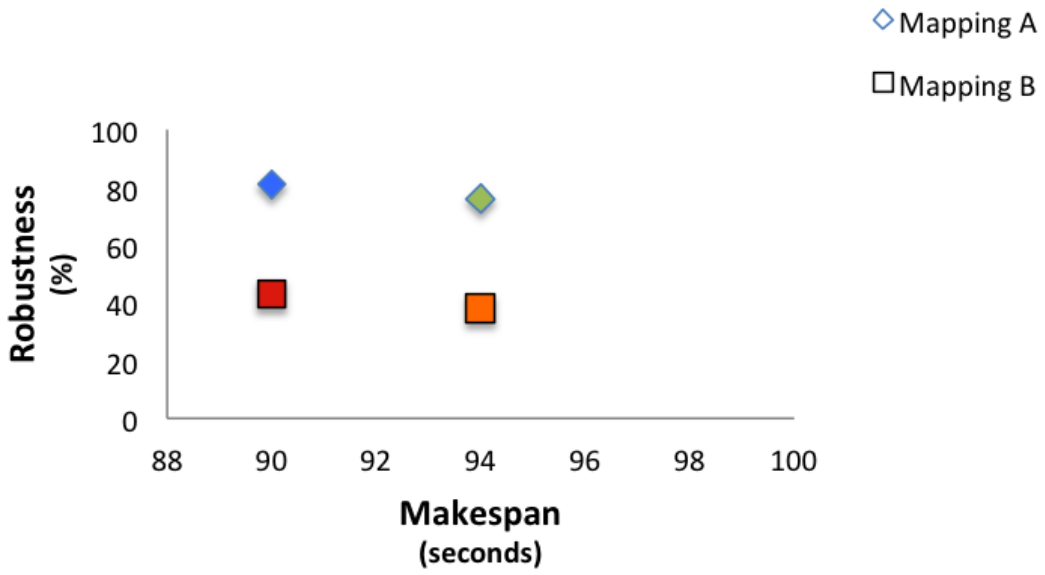

Figure 5.17: A comparison between the robustness values of the two resource allocations *Mapping A and Mapping B* delivering equal performance in terms of the system makespan.

5.2 Robustness evaluation case study: non-uniform workload variation across all applications

This modeling study is designed to evaluate the robustness of the mappings modeled in the previous case under a highly perturbed execution environment, where the sensors produce data at rapidly varying rates ($\lambda_1, \lambda_2, \lambda_3$) that may lead to a highly uncertain execution environment with entirely different sensor loads for every application. Like the previous case study, the execution of the two mappings, as given in Table 4.1, are modeled as CTMC processes using PEPA. The PEPA input file that defines the model for the two mappings is shown in Figure 4.2 and 4.3, respectively. Further, to complete the PEPA model, the rates $r1 \cdots r20$ and $p1 \cdots p20$, need to be calculated for solving the underlying Markovian model via a passage time analysis using the PEPA workbench to derive performance measures.

5.2.1 Deriving PEPA activity rates in ideal computing environment ($\hat{\lambda} = \lambda$)

The rates associated with the $compute_i$ activities, when there is no variation in the application workload, are represented by $r1 \cdots r20$. The calculation for the rate is given in Equation 5.1, where T_{ij} is the actual time to compute an application i on machine j, $\hat{\lambda}_i$ is calculated as a function of the varying sensor loads $\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$, and λ_i is calculated as a function of the initial sensor loads $\lambda_1, \lambda_2, \lambda_3$. The T_{ij} values of the 20 applications on the machines they are assigned to according to the two mappings are listed in Table 5.1 and 5.2, respectively [3]. T_{ij} is calculated as a product of the machine availability factor (η_i), which is the computational availability of the allocated machine j for executing application i, and the runtime workload for that application ($\hat{\lambda_i}$). The right most column of the table also lists the corresponding value of the ideal rate r_i at which the allocated machine executes application a_i. r_i is calculated as a ratio of the initial workload λ_i and T_{ij}. This signifies that the rate at which a machine will compute an assigned application is directly proportional to the workload value according to which the application was initially allocated to that machine and concurrently, is inversely affected by the actual computational time, which is dependent upon the actual (initial or varied) sensor loads during the execution of that application. In the ideal computing scenario, where $\hat{\lambda_i} = \lambda_i$, the value of r_i reduces to an inverse of the machine availability factor (η_i). The initial sensor load values are, $\lambda_1 = 962$, $\lambda_2 = 380$, and $\lambda_3 = 240$.

5.2.2 Deriving PEPA activity rates in perturbed computing environment ($\hat{\lambda} \neq \lambda$)

The rates associated with the $compute_i$ activities, in the presence of perturbations in the form of varying application workload, are represented by $p1 \cdots p20$. The calculation for the rate is given in Equation 5.2, where T_{ij} is the actual time to compute an application i on machine j, $\hat{\lambda_i}$ is calculated as a function of the varying sensor loads $\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$, and λ_i is calculated as a function of the initial sensor loads $\lambda_1, \lambda_2, \lambda_3$. The T_{ij} values of the 20 applications on the machines they are assigned to according to the two mappings are listed in Table 5.3 and 5.4, respectively [3]. T_{ij} is calculated as a product of the machine availability factor (η_i), which is the computational availability of the allocated machine j for executing application i, and the runtime workload for that application ($\hat{\lambda_i}$). The runtime application workload is calculated as a function of the three runtime sensor loads ($\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$). The right most column of the table also lists the corresponding value of the perturbed rate p_i at which the allocated machine executes application a_i. p_i is calculated as a ratio of the initial workload λ_i and T_{ij}. This signifies that the rate at which a machine will compute an assigned application is directly proportional to the workload value according to which the application was initially allocated to that machine and concurrently, is inversely affected by the actual computational time, which is dependent upon the actual (initial or varied) sensor loads during the execution of that application. In the perturbed computing scenario, $\hat{\lambda_i} \neq \lambda_i$. The initial sensor load values are, $\lambda_1 = 962$, $\lambda_2 = 380$, and $\lambda_3 = 240$. The runtime sensor load values are randomly sampled, for every application, from a mixture distribution hat consists a set of values generated from gamma, Gaussian, Erlang-K, and exponential distributions. The parameters of each of these distributions were derived as

functions of the shape parameter (k) and the scale parameter (θ) of the gamma distribution. The mixture distribution was created using the MATLAB statistics toolbox R2014b [73]. Therefore, for every application, three values of sensor loads $\hat{\lambda_1}, \hat{\lambda_2}, \hat{\lambda_3}$ were sampled from the mixture distribution. This also represents a case study for a resource allocation system that has a very high load imbalance factor at runtime.

5.2.3 Numerical Analysis and Validation of Performance Modeling of Resource Allocations using the PEPA Workbench

Once the ideal and the perturbed rates are calculated for completing the PEPA input files representing the performance models of the execution of applications on the allocated machines with respect to the two mappings, the PEPA models are compiled and solved using the PEPA workbench tool. The PEPA input files as shown in Figure 4.2 and 4.3, along with the rates r_i and p_i calculated using Tables 5.3, 5.4, 5.1, and 5.2, are compiled using the Eclipse Luna development tool [40] in a PEPA workbench framework, as shown in Figure 5.1. After the model compiles successfully according to the PEPA formalisms, the state space of the underlying mathematical Markovian model of the execution of the mappings is derived (shown in the screenshot in Figure 5.2).

The state space generated for the Markov model representing the execution of the two mappings consists of 10395 and 13475 number of states, respectively, of the CTMC processes representing the execution states of the applications and the allocated machines. Examples of activity diagrams resulting from the generated state space are illustrated in Figures 5.18 and 5.19. In the activity diagram, the rectangular boxes represent the different states of a PEPA component (or the underlying CTMC process). The arcs represent

the transitions between the states. The multiplicity of outgoing arcs (represented by two different colors) represent the possibility that a component may transition to either of the states at the end of one of the multiple outgoing arcs in its current state. For example, in Figure 5.3, the PEPA component M_3 has two outgoing arcs in its initial state. The two arcs represent the uncertainty that signifies the computation the application A_1 on the allocated machine M_3 in, either the *ideal* computing scenario with the rate $r_1 = 0.256$ or the *perturbed* computing scenario with the rate $p_1 = 0.054$. Every PEPA component is translated into its own activity diagram, modeling its evolution throughout the execution of the computational activities, similar to the activity diagrams shown in Figures 5.3 and 5.4.

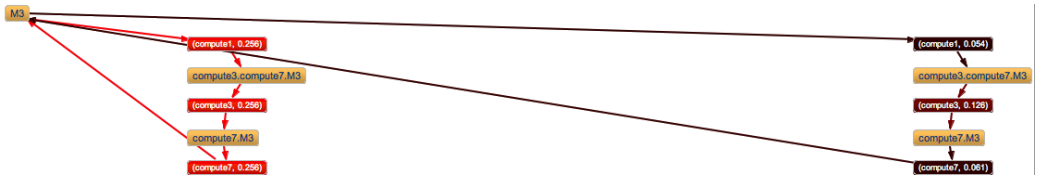

Figure 5.18: An activity diagram of the CTMC processes of the corresponding PEPA components in *Mapping A* for the modeling study in Case (ii).

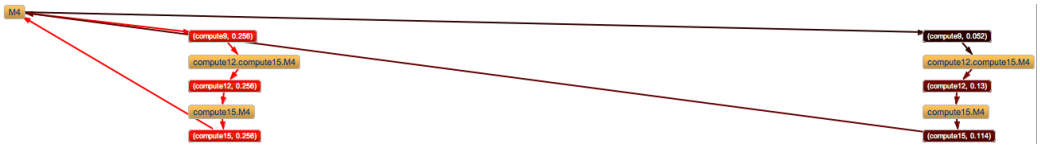

Figure 5.19: An activity diagram of the CTMC processes of the corresponding PEPA components in *Mapping B* for the modeling study in Case (ii).

Once the state space and the resulting activity diagrams of the components (CTMC processes) of the PEPA model are generated, the tool allows the modeler to specify the type of Markovian analysis that needs to be used for solving the generated Markov models to derive performance measures. As defined in Chapter 2, the PEPA workbench allows two types of Markovian analysis: (i) steady state analysis is used to solve the global balance equation (representing the state of equilibrium of the Markov model) using the infinitesimal generator matrix to calculate steady state probability values for every component to derive performance measures, such as, throughput and utilization, (ii) passage time analysis is used to solve the Markov models using the timing information associated with the activity rates to derive performance measures, such, as makespan and response time. In this case study, the performance feature of interest is *makespan*, therefore, passage time analysis is used to solve the underlying Markov models. Based on the knowledge that the modeler provides to the PEPA workbench in the form of PEPA formalism of the resource allocation system, the tool extracts the information from that knowledge in the form of initial state values, final state values, and the intermediate transitions. Using this information, the tool enables a passage time analysis of the Markov models for deriving the *makespan* as the passage time between the defined initial and final states. For example, as shown in Figure 5.5, the tool collects information from the PEPA model regarding the initial state(s) to be the state(s) associated with the $compute_1$ activity values and the final state(s) to be the state(s) associated with the $compute_8$ activity. The *start time* and the *stop time* values are specified the modeler and are often specified by the application user. For example, the *stop time* is analogous to the user specified makespan goal, β_i^{max}. In our modeling study,

the passage time analysis of the Markov models underlying the PEPA definitions of the two resource allocation mappings, yield CDFs of the machine finishing times, $F_i(M_j, \hat{\lambda}_i)$, as passage times between the states associated with the applications assigned to that machine. The CDFs of the finishing times of machines $M_1 \cdots M_5$ in the Markov model for `Mapping A` are shown in Figures 5.20 through 5.24, and for `Mapping B` are shown in Figures 5.25 through 5.29.

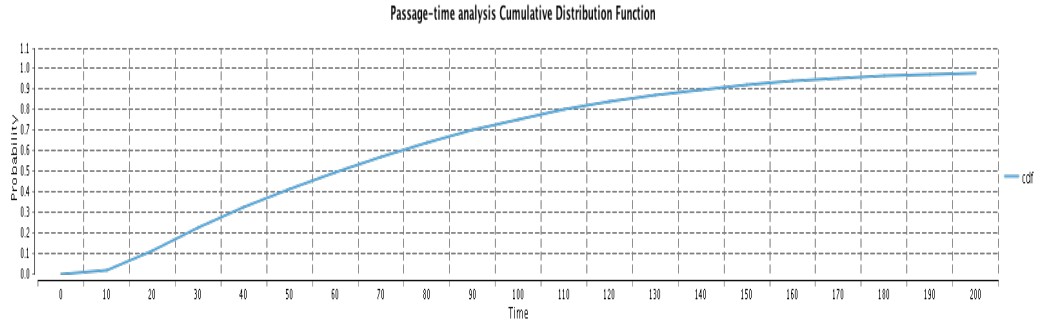

Figure 5.20: Cumulative distribution function (CDF) of the finishing time of machine M_1 for executing applications $A_5, A_9, A_{12}, A_{17}, A_{20}$ as given by *Mapping A* for the modeling study in Case (ii).

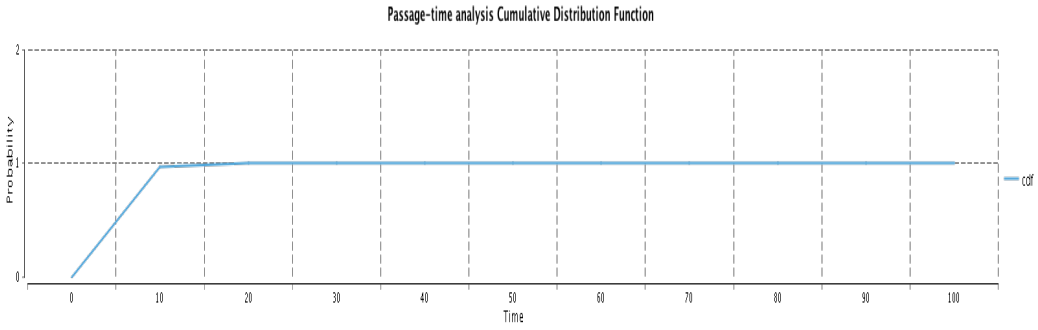

Figure 5.21: Cumulative distribution function (CDF) of the finishing time of machine M_2 for executing applications A_6, A_{16} as given by *Mapping A* for the modeling study in Case (ii).

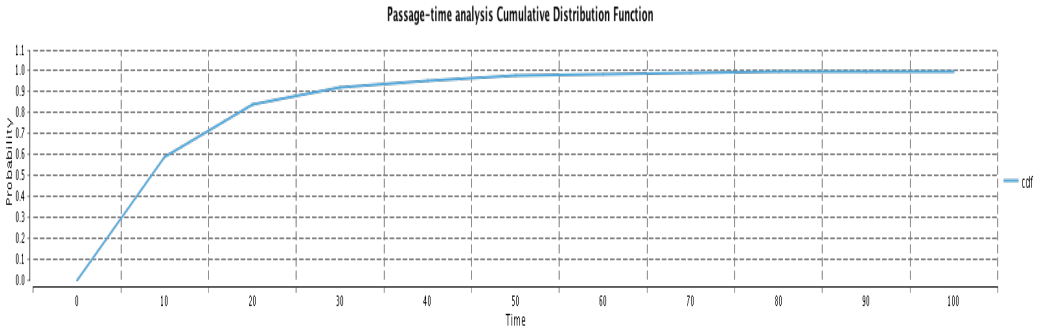

Figure 5.22: Cumulative distribution function (CDF) of the finishing time of machine M_3 for executing applications A_1, A_3, A_7 as given by *Mapping A* for the modeling study in Case (ii).

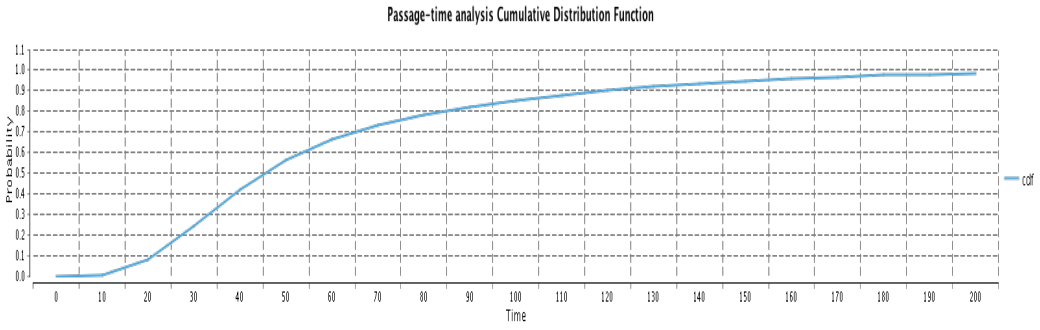

Figure 5.23: Cumulative distribution function (CDF) of the finishing time of machine M_4 for executing applications $A_2, A_4, A_{10}, A_{13}, A_{15}, A_{19}$ as given by *Mapping A* for the modeling study in Case (ii).

The robustness of the resource allocation defined by the performance modeling of the two mappings is formulated using Equation 4.1. Let ψ_A and ψ_B be the robustness values of the two mappings respectively. Similar to the modeling study in case (i), the makespan goal is set as $\beta_i^{max} = 45$ seconds. Then, the robustness of the two mappings is calculated using Equation 5.6 and 5.7. Both the two mappings yield a *makespan* of 90 seconds. However, they differ in their robustness values when the makespan goal is set to 45 seconds,

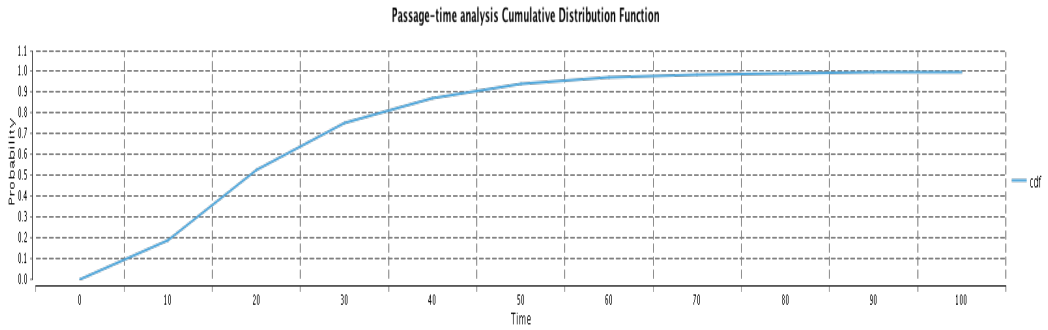

Figure 5.24: Cumulative distribution function (CDF) of the finishing time of machine M_5 for executing applications $A_8, A_{11}, A_{14}, A_{18}$ as given by *Mapping A* for the modeling study in Case (ii).

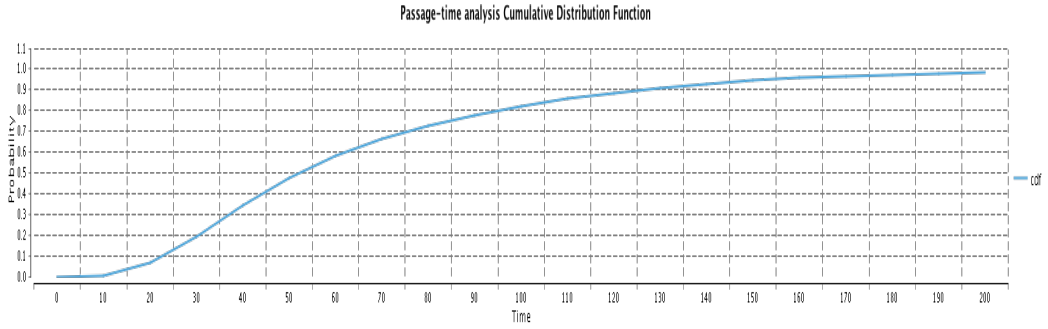

Figure 5.25: Cumulative distribution function (CDF) of the finishing time of machine M_1 for executing applications $A_3, A_4, A_5, A_{17}, A_{18}, A_{20}$ as given by *Mapping B* for the modeling study in Case (ii).

where $\psi_A = 35\%$ and $\psi_B = 41\%$. The difference in the robustness values of the two mappings that yield the same system makespan, reinstates the need for *robustness analysis* for selecting an initial mapping that can withstand the runtime perturbations in application and system parameters in a parallel computing environment. For example, in this modeling study, both the two mappings yield the same performance in the form of overall system makespan $\beta = 200$ seconds. However, `Mapping B` is a *more robust* choice for initial allocation in terms of achieving a set makespan goal in the presence of runtime perturbations,

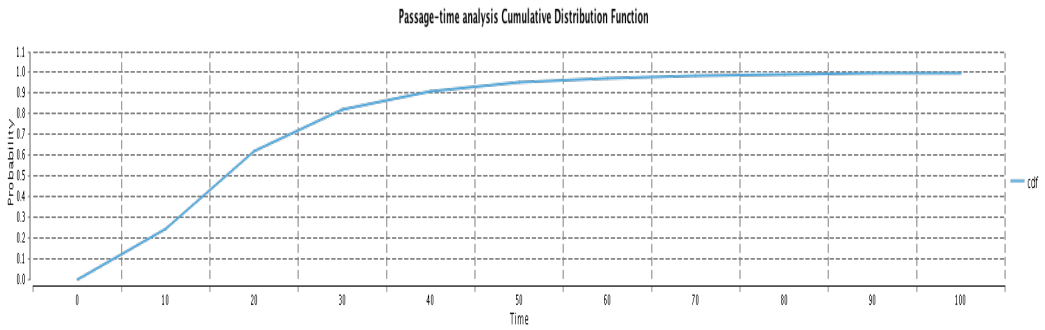

Figure 5.26: Cumulative distribution function (CDF) of the finishing time of machine M_2 for executing applications $A_2, A_{11}, A_{14}, A_{19}$ as given by *Mapping B* for the modeling study in Case (ii).

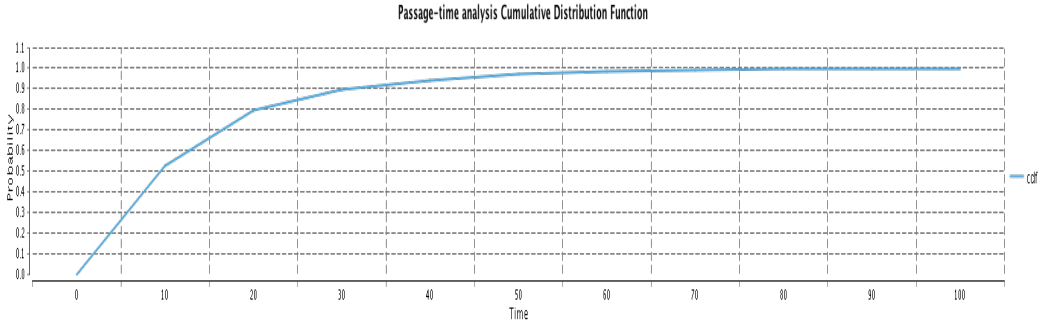

Figure 5.27: Cumulative distribution function (CDF) of the finishing time of machine M_3 for executing applications A_1, A_7, A_{13} as given by *Mapping B* for the modeling study in Case (ii).

such as workload variation. In addition, it is also evident that `Mapping A`, which was a better choice for the previous computational scenario with a low load imbalance factor, is a less robust choice for the computational scenario with a high load imbalance factor in this case study. A comparison of the robustness values associated with the two mappings in the two modeling studies, case (i) and (ii), is given in Figures 5.30 and 5.31. `Mapping A` is a more robust choice when there are equal workload variations across all applications, as shown in Figure 5.30. However, `Mapping B` is a more robust choice when there is

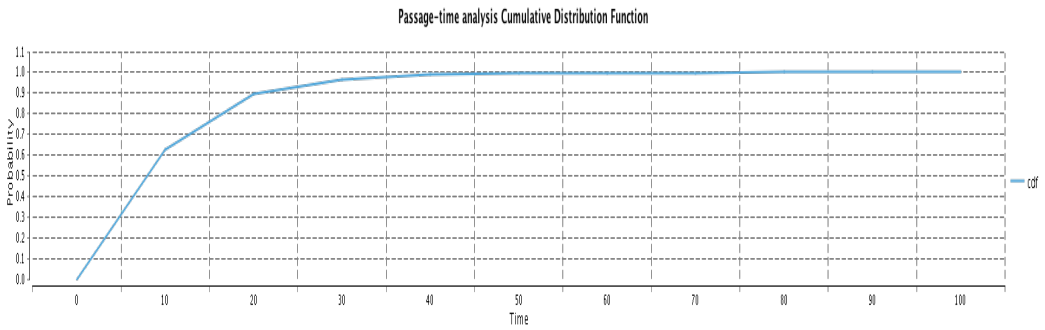

Figure 5.28: Cumulative distribution function (CDF) of the finishing time of machine M_4 for executing applications A_9, A_{12}, A_{15} as given by *Mapping B* for the modeling study in Case (ii).

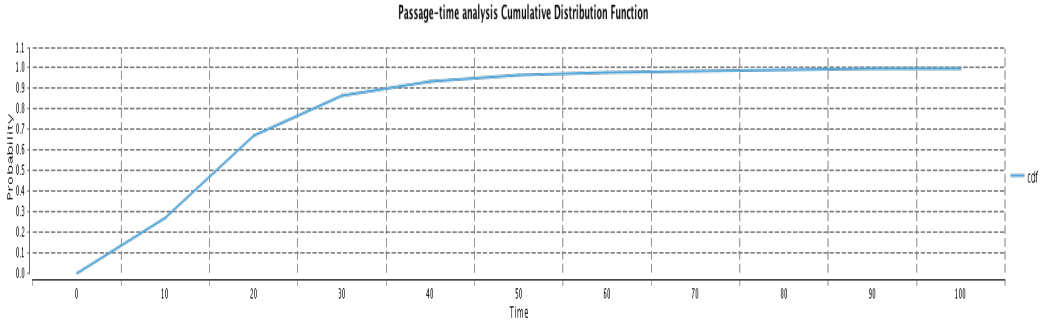

Figure 5.29: Cumulative distribution function (CDF) of the finishing time of machine M_5 for executing applications A_6, A_8, A_{10}, A_{16} as given by *Mapping B* for the modeling study in Case (ii).

non-uniform and highly random workload variation across all applications, as shown in Figure 5.31.

$$\psi_A = \min_{\forall i,j \; paired \; as \; in \; Mapping A} \Pr[F_i(M_j, \lambda_i) \leq 45] \tag{5.6}$$

$$\psi_B = \min_{\forall i,j \; paired \; as \; in \; Mapping B} \Pr[F_i(M_j, \lambda_i) \leq 45] \tag{5.7}$$

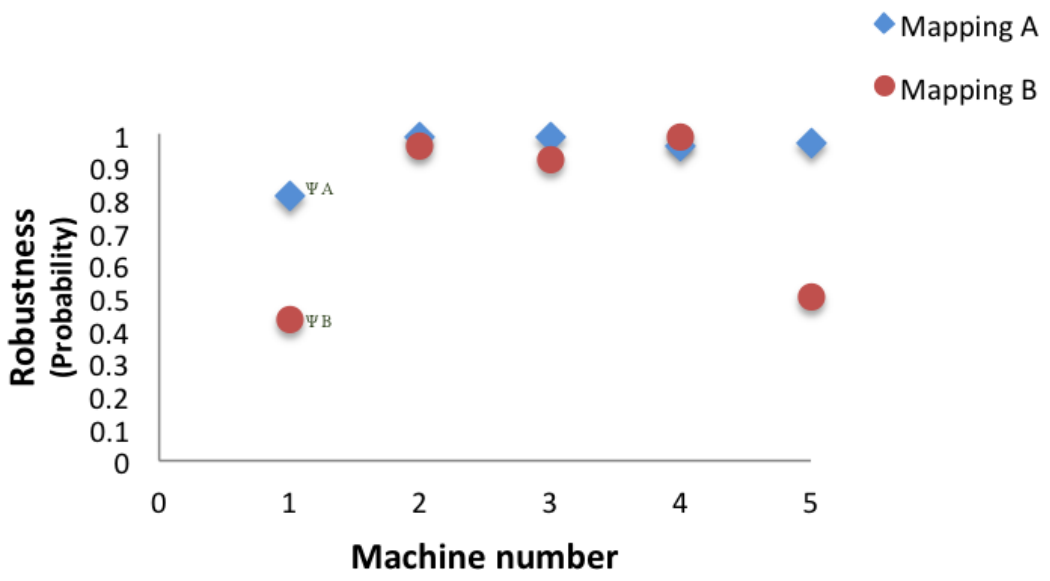

Figure 5.30: Case (i) probability values $s.t.$ $F_i(M_j, \lambda_i) \leq 45$ and the robustness.

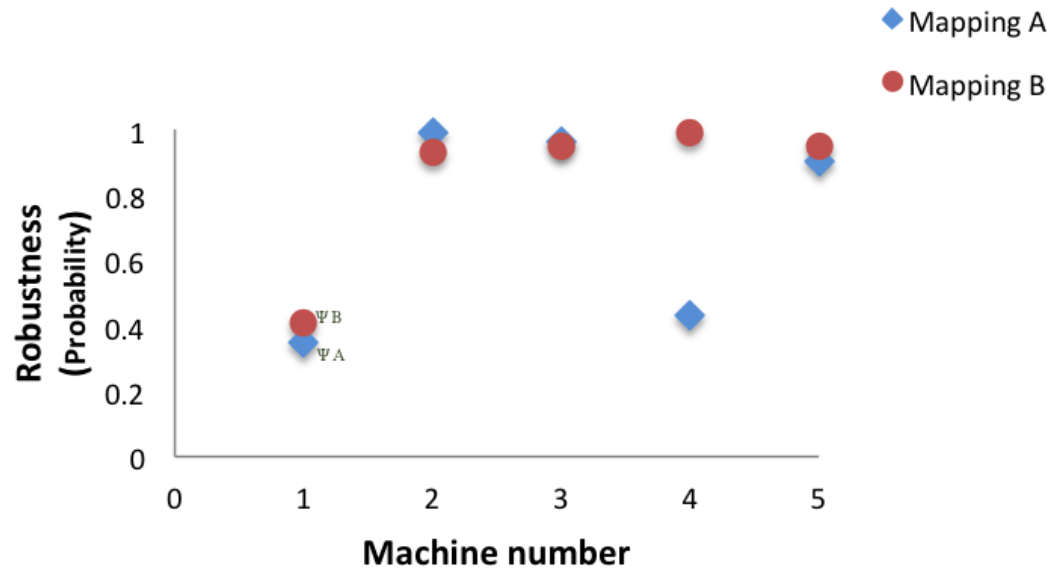

Figure 5.31: Case (ii) probability values $s.t.$ $F_i(M_j, \lambda_i) \leq 45$ and the robustness.

CHAPTER 6

BENEFITS, CONCLUSIONS, AND FUTURE WORK

6.1 Benefits of robustness analysis via analytical and numerical modeling of resource allocations

The results from the robustness analysis of resource allocations modeled analytically and solved numerically via PEPA can be used in the design phase of a parallel and distributed resource allocation system. The evaluation obtained from the PEPA models of robustness enables a more informed decision for selecting the most robust mapping from a set of initial mapping schemes. In addition, the analytical modeling enables selection of the most robust resource allocation from a set of resource allocations promising equal execution performance.

When compared to direct experiments and simulation, the analysis using analytical and numerical models is easier to reproduce, does not incur any setup or installation costs, does not impose any prerequisites for learning a simulation framework, and is not limited by the complexity of the underlying infrastructure or simulation libraries. Further, the use of PEPA modeling allows the system designer to seamlessly incorporate new system components and their behavior, for re-evaluation purposes. Stochastic process algebra models and the related numerical analysis provide a cost effective and low overhead analysis of robustness.

6.2 Conclusions and future work

Analytical and numerical models of resource allocations for parallel execution of applications, that have varying workload, on parallel and heterogeneous computing resources, has been developed in this work using a stochastic process algebra PEPA. Further, the performance evaluated from these models has been used for the robustness analysis of the resource allocations. The robustness obtained from the performance modeling of resource allocations has been validated against the results obtained from the robustness analysis performed in prior simulation experiments [3][6].

Novel contributions as well as the contributions that led to the research in this dissertation are listed below.

1. Performance modeling of resource allocations in parallel and distributed computing using PEPA.
2. Robustness evaluation using a passage time analysis (numerical analysis of Markovian models) of the developed performance models of resource allocations.
3. Robustness analysis of resource allocations w.r.t. equal variation in workload across all applications, and validating the robustness results of resource allocations obtained from performance modeling using PEPA with the robustness of the same resource allocations obtained via prior simulation experiments.
4. Robustness analysis of resource allocations w.r.t. the non-uniform variation in workload across all applications.
5. First implementation of the DLS methods in a simulation framework for a comparative analysis of the execution performance of these methods [108].
6. Study of the use of a model free machine learning (reinforcement learning) approach towards an automatic selection of the best DLS method for scheduling time-stepping scientific applications [16].
7. Formulation of robustness metrics for dynamic scheduling methods used in parallel computing systems and a study towards an online selection, of the most robust dynamic scheduling method, using machine learning techniques. The study of robustness of dynamic scheduling is applicable to a class of time-stepping scientific

applications and is a part of the foundation work on robustness that has led to the proposed thesis [107][110].

8. First implementation of a learning based methodology for an online prediction of the robustness of DLS methods using an artificial neural network [109].
9. A power-aware execution of scientific applications parallel and distributed computing systems using an existing model-based framework that combines the functionalities of DLS methods with a feedback limited look ahead controller [87].
10. A combined dual-stage framework for robust scheduling of scientific applications in heterogeneous environments with uncertain processor availability [32].

The proposed PEPA model for analyzing the robustness of a mapping in parallel and distributed computing can be useful for a predictive analysis in the design phase of a resource allocation system for selecting a robust initial mapping. Further, the proposed models can be extended with features to be useful for a runtime predictive analysis at predefined time samples during the execution of the application on the initially allocated resources. This will require an integration of the proposed PEPA models into a runtime scheduler/controller in a model-based framework, where the embedded models can be re-evaluated with minimal overhead when a system parameter changes at runtime.

REFERENCES

[1] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online Control for Self-Management in Computing Systems," *Proc. RTAS*, 2004, pp. 365–375.

[2] T. Abdelzaher, K. Shin, and N. Bhatti, "Performance guarantees for Web server end-systems: a control-theoretical approach," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 1, Jan 2002, pp. 80–96.

[3] S. Ali, *Robust resource allocation in dynamic distributed heterogeneous computing systems*, Purdue University, 2003.

[4] S. Ali, J.-K. Kim, H. J. Siegel, and A. A. Maciejewski, "Static Heuristics for Robust Resource Allocation of Continuously Executing Applications," *J. Parallel Distrib. Comput.*, vol. 68, no. 8, Aug. 2008, pp. 1070–1080.

[5] S. Ali, A. Maciejewski, and H. Siegel, *Perspectives on robust resource allocation for heterogeneous parallel systems*, Chapman & Hall/CRC Press, Boca Raton, FL, 2008.

[6] S. Ali, A. Maciejewski, H. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 7, 2004, pp. 630–641.

[7] S. Ali, A. Maciejewski, H. Siegel, and J.-K. Kim, "Robust resource allocation for sensor-actuator distributed computing systems," *Parallel Processing, 2004. ICPP 2004. International Conference on*, Aug 2004, pp. 178–185 vol.1.

[8] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, vol. 3, no. 3, 2000, pp. 195–208.

[9] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, 1967, AFIPS '67 (Spring), pp. 483–485.

[10] J. Baeten, N. D. o. C. S. Centrum voor Wiskunde en Informatica (Amsterdam, and R. van Glabbeek, *Merge and Termination in Process Algebra*, Centre for Mathematics and Computer Science, 1987.

[11] J. C. M. Baeten, "A brief history of process algebra," *Theor. Comput. Sci.*, vol. 335, no. 2-3, May 2005, pp. 131–146.

[12] M. Balasubramaniam, N. Sukhija, F. Ciorba, I. Banicescu, and S. Srivastava, "Towards the Scalability of Dynamic Loop Scheduling Techniques via Discrete Event Simulation," *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW-PDSEC, On CD-ROM), In Proceedings of The 2012 IEEE/ACM 26th International*, May 2012, pp. 1343–1351.

[13] I. Banicescu and R. L. Cariño, "Addressing the Stochastic Nature of Scientific Computations via Dynamic Loop Scheduling," *Electronic Transactions on Numerical Analysis - Special Issue on Combinatorial Scientific Computing*, vol. 21, 2005, pp. 66–80.

[14] I. Banicescu and R. L. Carino, "Addressing the stochastic nature of scientific computations via dynamic loop scheduling,", Electronic Transactions on Numerical Analysis 21:66-80, 2005.

[15] I. Banicescu, F. M. Ciorba, and R. L. Cariño, "Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems," *International Symposium on Parallel and Distributed Computing (ISPDC 2009)*, vol. 0, 2009, pp. 129–132.

[16] I. Banicescu, F. M. Ciorba, and S. Srivastava, "Performance Optimization of Scienti

c Applications using an Autonomic Computing Approach," *Scalable Computing and Communications: Theory and Practice*, A. Y. Z. Samee U. Khan and L. Wang, eds., John Wiley & Sons, Inc., 2013, pp. 437–466.

[17] I. Banicescu and S. Flynn Hummel, "Balancing processor loads and exploiting data locality in N-body simulations," *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, New York, NY, USA, 1995, p. 43, ACM.

[18] I. Banicescu and V. Velusamy, "Performance of Scheduling Scientific Applications with Adaptive Weighted Factoring," *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, Washington, DC, USA, 2001, p. 84, IEEE Computer Society.

[19] A. Benoit, M. Cole, S. Gilmore, and J. Hillston, "Evaluating the performance of skeleton-based high level parallel programs," *Computational Science-ICCS 2004*, Springer, 2004, pp. 289–296.

[20] A. Benoit, M. Cole, S. Gilmore, and J. Hillston, "Enhancing the effective utilisation of grid clusters by exploiting on-line performability analysis," *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, May 2005, vol. 1, pp. 317–324 Vol. 1.

[21] A. Benoit, M. Cole, S. Gilmore, and J. Hillston, "Scheduling skeleton-based grid applications using PEPA and NWS," *The Computer Journal*, vol. 48, no. 3, 2005, pp. 369–378.

[22] A. Benoit, M. Hakem, and Y. Robert, "Fault tolerant scheduling of precedence task graphs on heterogeneous platforms," *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–8.

[23] J. Bergstra, A. Ponse, and S. Smolka, *Handbook of Process Algebra*, Elsevier Science, 2001.

[24] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler," *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*. IEEE, 2003, pp. 344–351.

[25] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, vol. 14, no. 13, 2002, pp. 1175–1220.

[26] L. Blni, L. Boloni, and D. C. Marinescu, "Robust Scheduling of Metaprograms," *Journal of Scheduling*, vol. 5, 1998, pp. 395–412.

[27] R. L. Cariño and I. Banicescu, "Dynamic load balancing with adaptive factoring methods in scientific applications," *Journal of Supercomputing*, vol. 44, no. 1, Apr. 2008, pp. 41–63.

[28] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A Generic Framework for Large-Scale Distributed Experiments," *10th IEEE International Conference on Computer Modeling and Simulation*, Mar. 2008.

[29] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback–Feedforward Scheduling of Control Tasks," *Real-Time Syst.*, vol. 23, no. 1/2, 2002, pp. 25–53.

[30] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and standards for the evaluation of parallel job schedulers," *Job Scheduling Strategies for Parallel Processing*. Springer, 1999, pp. 67–90.

[31] W. chun Feng, X. Feng, and R. Ge, "Green Supercomputing Comes of Age," *IT Professional*, vol. 10, no. 1, 2008, pp. 17–23.

[32] F. Ciorba, T. Hansen, S. Srivastava, I. Banicescu, A. Maciejewski, and H. Siegel, "A Combined Dual-stage Framework for Robust Scheduling of Scientific Applications in Heterogeneous Environments with Uncertain Availability," *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW-HCW, On CD-ROM), In Proceedings of The 2012 IEEE/ACM 26th International*, May 2012, pp. 193–207.

[33] A. Clark, A. Duguid, and S. Gilmore, "Passage-End Analysis," *Computer Performance Engineering*, J. Bradley, ed., vol. 5652 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2009, pp. 110–115.

[34] E. Coffman and J. Bruno, *Computer and job-shop scheduling theory*, A Wiley-Interscience publication. Wiley, 1976.

[35] R. L. Daniels and J. E. Carrillo, "Beta-robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, vol. 29, no. 11, 1997, pp. 977–985.

[36] S. Dhandayuthapani, *Automatic Selection of Dynamic Loop Scheduling Algorithms for Load Balancing using Reinforcement Learning*, master's thesis, Mississippi State University, 2004.

[37] S. Dhandayuthapani, I. Banicescu, R. L. Cariño, E. Hansen, J. P. Pabico, and M. Rashid, "Automatic Selection of Loop Scheduling Algorithms Using Reinforcement Learning," *Challenges of Large Applications in Distributed Environments (CLADE 2005)*, 2005, pp. 87–94.

[38] Y. Dong, J. Chen, X. Yang, L. Deng, and X. Zhang, "Energy-Oriented OpenMP Parallel Loop Scheduling," *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, Washington, DC, USA, 2008, pp. 162–169, IEEE Computer Society.

[39] A. Dubey, R. Mehrotra, S. Abdelwahed, and A. Tantawi, "Performance modeling of distributed multi-tier enterprise systems," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, 2009, pp. 9–11.

[40] . Eclipse contributors 2000, "Eclipse documentation - Current Release Eclipse Luna,", 2013.

[41] T. Eguchi, F. Oba, and S. Toyooka, "A robust scheduling rule using a Neural Network in dynamically changing job-shop environments.," *IJMTM*, vol. 14, 2008, pp. 266–288.

[42] D. et. al., "The International Exascale Software Project roadmap," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, Feb. 2011, pp. 3–60.

[43] W. Feng, "Green Destiny + mpiBLAST = Bioinfomagic," *PARCO*, 2003, pp. 653–660.

[44] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Softw. Eng.*, vol. 15, no. 11, Nov. 1989, pp. 1427–1436.

[45] U. S. N. C. O. for Information Technology Research, Development, U. S. O. of Science, T. P. E. O. of the President, N. Science, and T. C. U. C. on Technology. High-End Computing Revitalization Task Force, *Federal Plan for High-end Computing: Report of the High-end Computing Revitalization Task Force (HECRTF).*, Executive Office of the President, Office of Science and Technology Policy, 2004.

[46] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, June 2007, pp. 835–848.

[47] R. Ge and K. Cameron, "Power-Aware Speedup," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, March 2007, pp. 1–10.

[48] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron, "CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters," *ICPP '07: Proceedings of the 2007 International Conference on Parallel Processing*, Washington, DC, USA, 2007, p. 18, IEEE Computer Society.

[49] S. Gertphol and V. Prasanna, "MIP formulation for robust resource allocation in dynamic real-time systems," *International Parallel and Distributed Processing Symposium, 2003. Proceedings.*, 2003, pp. 8 pp.–.

[50] S. Gertphol and V. Prasanna, "Iterative integer programming formulation for robust resource allocation in dynamic real-time systems," *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 118–.

[51] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel, "A Metric and Mixed-Integer-Programming-Based Approach for Resource Allocation in Dynamic Real-Time Systems," *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2002, IPDPS '02, IEEE Computer Society.

[52] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *Computer*, vol. 26, no. 6, June 1993, pp. 78–86.

[53] S. Gilmore and J. Hillston, "The PEPA workbench: A tool to support a process algebra-based approach to performance modelling," *Computer Performance Evaluation Modelling Techniques and Tools*, G. Haring and G. Kotsis, eds., vol. 794 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1994, pp. 353–368.

[54] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, 1988, pp. 532–533.

[55] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The Weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, Nov. 2009, pp. 10–18.

[56] R. Harrison, L. Zitzman, and G. Yoritomo, "High performance distributed computing program (HiPer-D)engineering testbed one (T1) report," *Naval Surface Warfare Center, Dahlgren, VA, Tech. Rep*, 1995.

[57] R. A. Hayden and J. T. Bradley, "A Fluid Analysis Framework for a Markovian Process Algebra," *Theor. Comput. Sci.*, vol. 411, no. 22-24, May 2010, pp. 2260–2297.

[58] R. A. Hayden, J. T. Bradley, and A. Clark, "Performance Specification and Evaluation with Unified Stochastic Probes and Fluid Analysis," *IEEE Trans. Softw. Eng.*, vol. 39, no. 1, Jan. 2013, pp. 97–118.

[59] J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[60] J. Hillston, "Tuning Systems: From Composition to Performance," *The Computer Journal*, vol. 48, no. 4, May 2005, pp. 385–400.

[61] C. A. R. Hoare, *Communicating sequential processes*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[62] C.-H. Hsu and W.-C. Feng, "Effective dynamic voltage scaling through CPU-boundedness detection," *In Workshop on Power Aware Computing Systems*, 2004, pp. 135–149.

[63] C.-h. Hsu and W.-c. Feng, "A Power-Aware Run-Time System for High-Performance Computing," *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2005, p. 1, IEEE Computer Society.

[64] S. F. Hummel, J. Schmidt, R. Uma, and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring," *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*. ACM, 1996, pp. 318–328.

[65] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring," *SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, New York, NY, USA, 1996, pp. 318–328, ACM.

[66] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A method for scheduling parallel loops," *Communications of the ACM*, vol. 35, no. 8, 1992, pp. 90–101.

[67] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: a method for scheduling parallel loops," *Commun. ACM*, vol. 35, no. 8, 1992, pp. 90–101.

[68] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *J. ACM*, vol. 24, no. 2, Apr. 1977, pp. 280–289.

[69] M. A. Iverson, F. Özgüner, and L. Potter, "Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment," *IEEE Trans. Comput.*, vol. 48, no. 12, Dec. 1999, pp. 1374–1379.

[70] R. Jain, *The art of computer systems performance analysis*, John Wiley & Sons, 2008.

[71] D. Janovy, J. Smith, H. Siegel, and A. Maciejewski, "Models and Heuristics for Robust Resource Allocation in Parallel and Distributed Computing Systems," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, March 2007, pp. 1–5.

[72] M. T. Jensen, "Improving Robustness and Flexibility of Tardiness and Total Flowtime Job Shops Using Robustness Measures," *Journal of Applied Soft Computing*, vol. 1, 2001, pp. 35–52.

[73] B. Jones, *MATLAB: Statistics Toolbox; User's Guide*, MathWorks, 1997.

[74] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, July 1998, pp. 42–51.

[75] N. Kandasamy, S. Abdelwahed, and M. Khandekar, "A hierarchical optimization framework for autonomic performance management of distributed computing systems," *Proc. 26th IEEE Int'l Conf. Distributed Computing Systems (ICDCS)*, 2006.

[76] D. Klusáček and H. Rudová, "Alea 2 – Job Scheduling Simulator," *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. 2010, ICST.

[77] P. Koopman, K. DeVale, and J. DeVale, "INTERFACE ROBUSTNESS TESTING: EXPERIENCES AND LESSONS LEARNED FROM the Ballista project,", 2008.

[78] C. P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors," *Software Engineering, IEEE Transactions on*, , no. 10, 1985, pp. 1001–1016.

[79] C. P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Trans. Softw. Eng.*, vol. 11, no. 10, 1985, pp. 1001–1016.

[80] D. Kusic, N. Kandasamy, and G. Jiang, "Approximation Modeling for the Online Performance Management of Distributed Computing Systems," *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, Washington, DC, USA, 2007, p. 23, IEEE Computer Society.

[81] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[82] J. V. Leon, D. S. Wu, and R. H. Storer, "Robustness Measures and Robust Scheduling for Job Shops," *IIE Transactions*, vol. 26, no. 5, 1994, pp. 32–43.

[83] N. Lopez-Benitez and J.-Y. Hyon, "Simulation of task graph systems in heterogeneous computing environments," *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, 1999, pp. 112–124.

[84] C. Lu, G. A. Alvarez, and J. Wilkes, "Aqueduct: Online Data Migration with Performance Guarantees," *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Berkeley, CA, USA, 2002, p. 21, USENIX Association.

[85] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, 2003, pp. 1105–1122.

[86] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *In Encyclopedia of Electrical and Electronics Engineering*. 1999, pp. 679–690, John Wiley.

[87] R. Mehrotra, I. Banicescu, and S. Srivastava, "A Utility Based Power-Aware Autonomic Approach for Running Scientific Applications," *In Proceedings of The 2012 IEEE/ACM 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW-PDSEC, On CD-ROM),*, May 2012, pp. 1457–1466.

[88] R. Mehrotra, A. Dubey, S. Abdelwahed, and W. Monceaux, "Large Scale Monitoring and Online Analysis in a Distributed Virtualized Environment," *Engineering of Autonomic and Autonomous Systems, IEEE International Workshop on*, vol. 0, 2011, pp. 1–9.

[89] R. Mehrotra, A. Dubey, S. Abdelwahed, and A. Tantawi, "Integrated Monitoring and Control for Performance Management of Distributed Enterprise Systems," *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, vol. 0, 2010, pp. 424–426.

[90] A. Mehta, J. Smith, H. Siegel, A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness," *The Journal of Supercomputing*, vol. 42, no. 1, 2007, pp. 33–58.

[91] J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *IEEE Trans. Comput.*, vol. 29, no. 8, Aug. 1980, pp. 720–731.

[92] P. V. Mieghem, *Performance Analysis of Communications Networks and Systems*, Cambridge University Press, New York, NY, USA, 2005.

[93] R. Milner, *Communication and Concurrency*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[94] G. D. Plotkin, "The origins of structural operational semantics," *The Journal of Logic and Algebraic Programming*, vol. 6061, 2004, pp. 3 – 15.

[95] C. D. Polychronopoulos and D. J. Kuck, "Guided Self-scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. Comput.*, vol. 36, no. 12, Dec. 1987, pp. 1425–1439.

[96] M. Rashid, I. Banicescu, and R. L. Cariño, "Investigating a dynamic loop scheduling with reinforcement learning approach to load balancing scientific applications," *7th IEEE Int. Symposium on Parallel and Distributed Computing (ISPDC 2008)*, 2008, vol. 0, pp. 123–130.

[97] E. Rich and K. Knight, *Artificial Intelligence*, McGraw-Hill Science/Engineering/Math, December 1990.

[98] V. Shestak, E. K. P. Chong, H. J. Siegel, A. A. Maciejewski, L. Benmohamed, I.-J. Wang, and R. Daley, "A Hybrid Branch-and-Bound and Evolutionary Approach for Allocating Strings of Applications to Heterogeneous Distributed Computing Systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, Apr. 2008, pp. 410–426.

[99] V. Shestak, H. Siegel, A. Maciejewski, and S. Ali, "The Robustness of Resource Allocations in Parallel and Distributed Computing Systems," *Architecture of Computing Systems - ARCS 2006*, W. Grass, B. Sick, and K. Waldschmidt, eds., vol. 3894 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 17–30.

[100] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Iterative algorithms for stochastically robust static resource allocation in periodic sensor driven clusters," *in: 8th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006*, 2006, pp. 166–174.

[101] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *J. Parallel Distrib. Comput.*, vol. 68, no. 8, Aug. 2008, pp. 1157–1173.

[102] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic Robustness Metric and Its Use for Static Resource Allocations," *J. Parallel Distrib. Comput.*, vol. 68, no. 8, Aug. 2008, pp. 1157–1173.

[103] V. Shestak, J. Smith, H. Siegel, and A. Maciejewski, "A Stochastic Approach to Measuring the Robustness of Resource Allocations in Distributed Systems," *Parallel Processing, 2006. ICPP 2006. International Conference on*, Aug 2006, pp. 459–470.

[104] V. Shestak, J. Smith, R. Uml, J. Hale, P. Moranville, A. A. Maciejewski, and H. J. Siegel, "Greedy approaches to static stochastic robust resource allocation for periodic sensor driven distributed systems," *In Proceedings the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA06*, 2006, pp. 3–9.

[105] B. J. Smith, "Architecture and applications of the HEP multiprocessor computer system," *SPIE - Real-Time Signal Processing IV*, 1981, pp. 241–248.

[106] J. Smith, H. J. Siegel, and A. A. Maciejewski, "Robust Resource Allocation in Heterogeneous Parallel and Distributed Computing Systems," *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[107] S. Srivastava, I. Banicescu, and F. Ciorba, "Investigating the Robustness of Adaptive Dynamic Loop Scheduling on Heterogeneous Computing Systems," *In Proceedings of The 2010 IEEE/ACM International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW-PDSEC, On CD-ROM)*, April 2010, pp. 1–8.

[108] S. Srivastava, I. Banicescu, F. Ciorba, and W. Nagel, "Enhancing the Functionality of a GridSim-Based Scheduler for Effective Use with Large-Scale Scientific Applications," *Parallel and Distributed Computing (ISPDC, On CD-ROM), In Proceedings of The 2011 10th IEEE International Symposium on*, July 2011, pp. 86–93.

[109] S. Srivastava, B. Malone, N. Sukhija, I. Banicescu, and F. Ciorba, "Predicting the Flexibility of Dynamic Loop Scheduling Using an Artificial Neural Network," *Parallel and Distributed Computing (ISPDC, On CD-ROM), In Proceedings of The 2013 IEEE 12th International Symposium on*, June 2013, pp. 3–10.

[110] S. Srivastava, N. Sukhija, I. Banicescu, and F. Ciorba, "Analyzing the Robustness of Dynamic Loop Scheduling for Heterogeneous Computing Systems," *Parallel and Distributed Computing (ISPDC), In Proceedings of The 2012 11th IEEE International Symposium on*, June 2012, pp. 156–163.

[111] P. Sugavanam, H. Siegel, A. A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. Pippin, "Robust static allocation of resources for independent tasks under makespan and dollar cost constraints," *Journal of Parallel and Distributed Computing*, vol. 67, no. 4, 2007, pp. 400 – 416.

[112] N. Sukhija, I. Banicescu, S. Srivastava, and F. Ciorba, "Evaluating the Flexibility of Dynamic Loop Scheduling on Heterogeneous Systems in the Presence of Fluctuating Load Using SimGrid," *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW-PDSEC, On CD-ROM), In Proceedings of The 2013 IEEE/ACM 27th International*, May 2013, pp. 1429–1438.

[113] T. H. Tzen and L. M. Ni, "Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, 1993, pp. 87–98.

[114] V. L. Wallace and R. S. Rosenberg, "Markovian models and numerical analysis of computer system behavior," *Proceedings of the April 26-28, 1966, Spring joint computer conference*. ACM, 1966, pp. 141–148.

[115] C. J. C. H. Watkins and P. Dyan, "Q-Learning," *Machine Learning*, vol. 8, no. 3–4, May 1992, pp. 279–292.

[116] F. Xia and Y. Sun, "Neural Network Based Feedback Scheduling of Multitasking Control Systems," *Proc. KES2005, Lecture Notes in Computer Science,*. 2005, pp. 237–246, Springer-Verlag.

APPENDIX A

ADDITIONAL WORK RELATED TO DISSERTATION RESEARCH

The work presented in this chapter is related to the study of a power aware execution of scientific applications on parallel and distributed computing systems. The study was conducted as a part of the project related to the NSF Center for Cloud and Autonomic Computing at Mississippi State University. My contribution to the project was related to providing dynamic load balancing features via a number of dynamic scheduling methods that were used to schedule the parallel and independent tasks within the scientific applications on the allocated machines with power tuning capabilities. An article was delivered as a part of the study and was published in a renowned IEEE conference proceeding [87].

A.1 A Utility Based Power-Aware Autonomic Approach for Running Scientific Applications

The primary objective for designing high performance computing (HPC) systems is performance - to solve scientific problems in minimum time with efficient or maximum utilization of the allocated computing resources. The efficient utilization of the allocated computing resources is achieved by following two objectives: (i) decreasing the computational and interprocess communication overheads, and (ii) ensuring that computing resources are effectively utilized in doing useful work at their peak performance all the time. With the recent advancements in the HPC system size and the processing power of computing nodes led to a significant increase in the power consumption. A study commissioned by the U.S. Environmental Protection Agency estimates that the worldwide power consumed by servers increased by a factor of two between 2000 and 2006 worldwide. In addition to this, an increase in power consumption results in increased temperature, which in turn translates into increased heat dissipation issues, resulting in increased system failure rate

that leads to downtime penalty for the service providers in extremely high values. According to Arrhenius' equation when applied to HPC hardware, each 10 degree increase in system's temperature doubles the system failure rate [31]. These HPC systems with enormous heat dissipation need aggressive cooling systems. These additional deployment of aggressive cooling systems contribute even more to the power consumption of the infrastructure resulting into an increased operating cost. Another solution to the heat dissipation problem is to increase the space between different computing nodes. However, this approach results in high infrastructure cost due to larger space used for fewer computing nodes. Both of these solutions ignore the performance vs. operating cost and performance vs. space cost metrics of the system, which have become significant to the service providers in the current scenario.

Solutions to the increased concerns of outrageous amounts of power consumption in HPC environments have led to the development of *Green Supercomputing* or *Energy Efficient Computing* technologies. Green Supercomputing provides two major approaches for minimizing power consumption in HPC environments while maintaining the desired performance. These approaches are: Low-Power and Power-Aware.

In Low-Power approach, the HPC system consists of many low power nodes which can be configured for a supercomputing environment. *Green Destiny* was the first low power supercomputer developed at Los Alamos National Laboratory [43]. It consists of 240 computing nodes working at the rate of 240 gigaflops on a linux-based cluster. It fits into a six square feet surface, consumes only 3.2 kW of power, and extra cooling or space is not required. The design of this supercomputer led to a breakthrough in the technology

of HPC environments, and shifted the focus towards efficiency, availability, and reliability of the computing systems in addition to speed.

In a Power-Aware approach, the HPC system is considered to be *aware* of the power requirements of the system executing scientific applications. In these systems, the power can be dynamically adjusted based on the demands of the application, while maintaining the performance at desired levels [48, 47, 63]. In most HPC systems, the *power aware* functionality is achieved through Dynamic Voltage and Frequency Scaling (DVFS). The power consumption by a processor directly depends upon the frequency and the square of the CPU supplied voltage. The frequency or the voltage across the processor can be decreased when CPU is not doing any or much useful work. This approach translates into considerable power savings.

In general, scientific applications are large, highly irregular, computational intensive, and data parallel. Often, one of the major difficulties in achieving performance objectives when running scientific applications in heterogeneous environments is their stochastic behavior [14]. This stochastic behavior results in severe load imbalance, which degrades the performance of the overall HPC system to a great extent, and becomes a potential threat for missing a pre-specified execution deadline. Moreover, the variability of the resource consumption behavior of other applications running on the same computing node (on which a parallel application is running) represents an additional overhead for meeting the deadline in HPC system. Due to these issues, a runtime monitoring and corrective strategy is required that can reallocate or reconfigure the computational resources.

Parallel loops are the dominant source of parallelism in scientific computing applications. For minimizing the computation time of an application, the loop iterations need to be executed efficiently over the network of computing nodes. In recent years, various loop scheduling techniques based on probabilistic analyses have been developed and applied to effectively allocate these iterations to different computing nodes and improve performance of applications via load balancing. Details regarding these techniques can be found in the related literature on loop scheduling, and are also summarized in [14]. However, these loop scheduling algorithms do not consider power-awareness, reallocation of computing resources, and application deadline.

In this paper, a *control theoretic, model-based, and power-aware* approach for parallel loop execution is presented, where system performance and power requirements can be adjusted dynamically, while maintaining the predefined quality of service (QoS) goals in the presence of perturbations such as variations in the availability of computational resources. In this approach, the target turnaround time for the application is identified or supplied and guides the appropriate adjustment of the voltage and frequency. The adjustment is made at the computational resource to ensure that the application finishes execution by the prespecified deadline (within an acceptable or chosen tolerance value) is performed with the help of a control theoretic approach. This approach also ensures the load-balanced execution of the application feature in the computing environment because all of the processing nodes finish execution simultaneously within an acceptable tolerance value. The general framework of this work considers multiple applications (each one being deadline driven) executing over a set of computational resources, where each application has its own set

of dedicated resources. This approach is autonomic, performance directed, dynamically controlled, and independent of the execution of the execution of application.

The paper is organized as follows. The background knowledge and related works are presented in Section A.2, while the proposed approach is described in Section A.3. The simulation setup and results are discussed in Section A.4, while benefits of the approach are highlighted in Section A.5. Finally, conclusions and future work are presented in Section A.6.

A.2 Background

In this section past and ongoing research efforts, such as: approaches based on power-aware, loop scheduling, and control theory for HPC systems are discussed.

A.2.1 Power-Aware Approaches with DVFS

Power-Aware computing has recently gained attention of the research communities in HPC systems for the purpose of lowering the power consumption and for increasing the system availability and reliability. The proposed power-aware approaches attempt to model the power consumption pattern of the scientific application, or that of the entire HPC system, based on the application and/or system performance. It is always recommended to minimize the power consumption of the HPC system with minimal or no impact on system performance.

An effort to minimize the power consumption of a HPC system through identifying the different execution phases (memory access, I/O access, and system idle) and their performance requirements while executing scientific applications is highlighted in [48]. In this

approach, a minimum frequency for each execution phase is determined and applied to the system for achieving the desired system performance. Another approach [47], presents a speedup model to minimize the power consumption while maintaining the similar application performance through identifying the relationship between parallel overhead and power requirement of an application via their impact on execution time. Through this model, the parallel overhead and the power-aware performance can be predicted over several system configurations. The approach shown in [62] describes a DVFS algorithm that detects the level of CPU-Boundness of the scientific application at runtime via dynamic regression and adjusts the CPU frequency accordingly. Another approach of utilizing the multiple power-performance state is presented in [46], and shows that a power scalable system can save significant amount of energy with negligible time penalty while maintaining the system QoS parameters.

A.2.2 Loop Scheduling

In the past years, extensive work has been performed in academia and industry to improve the performance of scientific applications through achieving load balancing via scheduling of parallel loops present in the applications. There are two primary methods of loop scheduling: static loop scheduling and dynamic loop scheduling (DLS).

In case of static loop scheduling, parallel loops are assigned to multiple processing elements (computing nodes) in blocks of fix sizes. The blocks contain iteration of variable execution times, thus causing load imbalance among processors.

The approach of dynamic loop scheduling assigns the parallel loop iterations at runtime one by one in a group of iterations or chunks. Each processing element keeps executing the iterates until all of them are finished. The simplest of these schemes is Self-Scheduling [105] where each processing element executes one iteration of the loop at a time until it finishes all the iterations. This scheme achieves perfect load balancing among the processing elements at the cost of high synchronization overhead. Other approaches presented in [79, 113, 67, 17] consider the profile of the integrations, availability of processing elements, chunk size, or locality of the data elements while assigning the iterations to the processors at runtime.

In the past, the loop scheduling methods addressed load imbalance and did not take into account the fact that the application performance may vary both due to algorithmic characteristic of the application and system related issues (interference by other applications running by the operating system). However, recent techniques are based on probabilistic analysis and take into account these factors when assigning the iterations to processors at runtime [79, 113, 67, 17].

There are also several adaptive approaches offering better performances described in [65, 18]. These approaches consider processor speed and performance while distributing the jobs resulting into better results. In [65], the ratio of job distribution is determined based on the relative processor speed. However, it does not take into account the fact that the processor speed may vary due to algorithmic characteristic of the application or due to system related issues (interference by the other applications running on the operating system). A similar kind of approach is described in [18], where the distribution of loop

iterations depends upon the processor performance at runtime. Each processor is assigned a weight based on its performance at the last sample period and receives a chunk of appropriate size, such that all processors finish at the same time with high probability. These weights are dynamically computed every time when a processor is allocated a chunk of loop iterations.

A.2.3 DVFS Based Loop Scheduling

DVFS based loop scheduling techniques take advantage of multiple power modes of the processing elements to control the load imbalance issue during the execution and to save the power consumption of the executing system by lowering down the speed or shutting down the idle processing element. These approaches utilize the notion of DVFS in the ways described below:

In Shut Down Based Techniques, fixed chunks of loop iterations are assigned to each processor within a group of processors at the beginning, and they start executing their assignment. As soon as a processor finishes the execution of its assigned iterations, the system assigns to it the minimum frequency (standby). This scheme does not ensure load balancing (simultaneous completion of the execution of iterations by processors). However, it offers minimal power consumption by forcing idle processors in the low power mode.

In the DVFS with Static Scheduling scheme, fixed size chunks of loop iterations are assigned to each processor within a group of the processors before they start executing their

assignments. Each processor is assigned the optimal frequency of system operation with respect to the execution time taken by the slowest processor (at the beginning) and then proceeds with the execution of its fixed size assigned chunks [38]. This approach needs prior information regarding the execution time of loop iterations at different processors to select the slowest processor.

In case of DVFS with dynamic scheduling, to minimize the power consumption, the chunk of loop iterations can be assigned to the group of processors at runtime and the processor that finishes before the slowest processor will be kept at minimum frequency. This approach is an extension of shut down based techniques with dynamic loop scheduling.

A.2.4 Elements of Control Theory

Control theory concepts offer a powerful ground to investigate various resource management, uncertain changes, and system disturbance issues. Recently, control theoretic approaches have successfully been applied to selected resource management problems including task scheduling [29], bandwidth allocation, QoS adaptation in web servers [2, 88], multi-tier websites [84, 89], load balancing in e-mail and file servers [84], and processor power management [80]. Control theory provides a taxonomy to design an automated, self-managed and effective resource management or partition scheme by continuous monitoring on the system states, changes in the environmental input, and system response to these changes. This scheme ensures that the system is always operating in the region of safe operational states, while maintaining the QoS demands of the service provider.

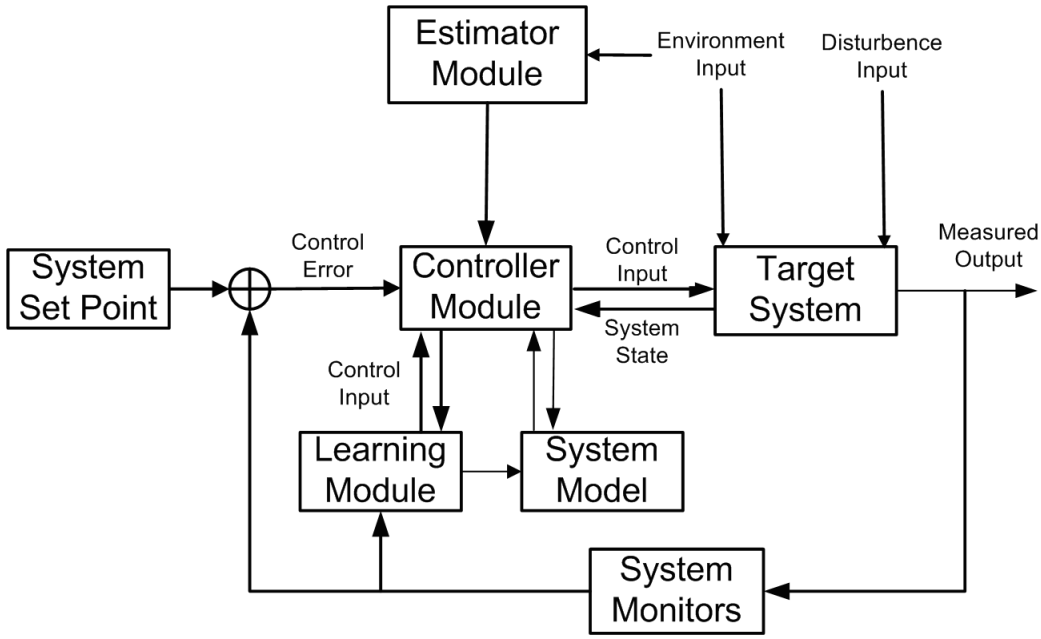

Figure A.1: Structure of a Control System.

A typical control system consists of the components shown in Figure A.1. The *System Set Point* is the desired state of the system considering in consideration that a system tries to achieve during its operation. The *Control Error* indicates the difference between the desired system set point and the measured output during system operation. The *Control Inputs* are the set of system parameters which are applied dynamically to the system dynamically for changing the performance level. The *Controller Module* monitors the measured output and provides the optimal combination of different control inputs to achieve the desired set point. The *Estimator Module* estimates the unknown parameters for the system based upon the previous history using statistical methods. The *Disturbance input* can be considered as the environment input that affects the system performance. The *Target system* is the system in consideration, while the *System Model* is the mathematical model

of the system, which defines the relationship between its input and output variables. The *Learning Module* collects the output through the monitor and extracts information based on statistical methods. Typically, the *System State* defines the relationship between the control or the input variables, and the performance parameters of the system.

A.3 Proposed Approach

A generic control framework is designed in [1, 75] for the performance management problem in computing systems. For each time interval, the control function tries to optimize the multi-dimensional QoS objective with respect to the cost incurred while using the computational resources. In this framework, control actions are taken based on continuous observation of the system states and environmental input variation are predicted by a mathematical system model for achieving predefined QoS objectives, in terms of execution time and minimum power consumption.

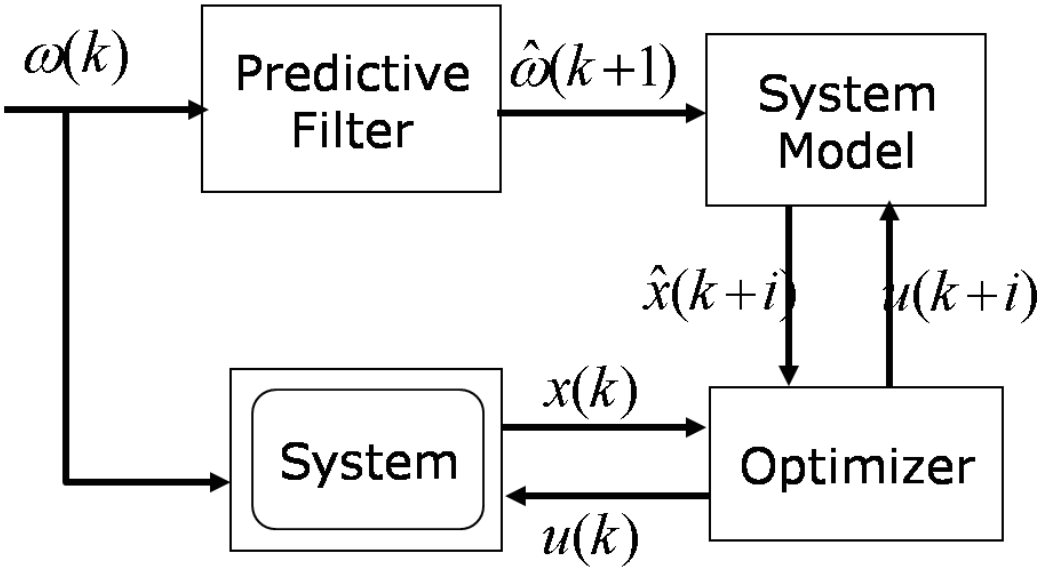

Figure A.2: Online Controller Architecture.

Prior work on scheduling large numbers of loop iterations (N) in a multiprocessor environment with P processors through a static scheduling approach is presented in [38]. In this approach, equal amounts to loop iterations ($\frac{N}{P}$) are assigned to each of the processors at the beginning of execution. These processors are assigned frequencies(f_i, where $i \in (1...P)$) in terms of their relative speed of execution, such that these processors will finish the execution at the same time with minimum time and minimum load imbalance [38]. However, this approach ignores the possibility of variation in the computational resource (i.e. CPU cycles) availability for scheduling the loop iterations. This variation in the availability of computational resources is caused by the use of the same resources by other applications (i.e. I/O or OS). In extreme cases, the perturbation in the available CPU at different processing nodes may result in a severe load imbalance among them, and in turn result in missing the execution deadline. Thus, an effective monitoring technique should be used to reconfigure the computational resources to achieve the predefined deadline.

A solution to the above problem employs an effective monitoring technique within the proposed control framework that keeps track of the total number of completed iterates by each processor after each sample time (T), and re-adjusts the assigned CPU frequencies in a way that these processors finish within the deadline T_d while consuming minimum power. The increment in CPU frequencies ensures that the loop execution environment receives sufficient computational power (CPU cycles) even in the presence of overhead applications that takes some percentage of the total CPU cycles. The proposed approach lowers the CPU frequency in case of low overhead in CPU availability at runtime after

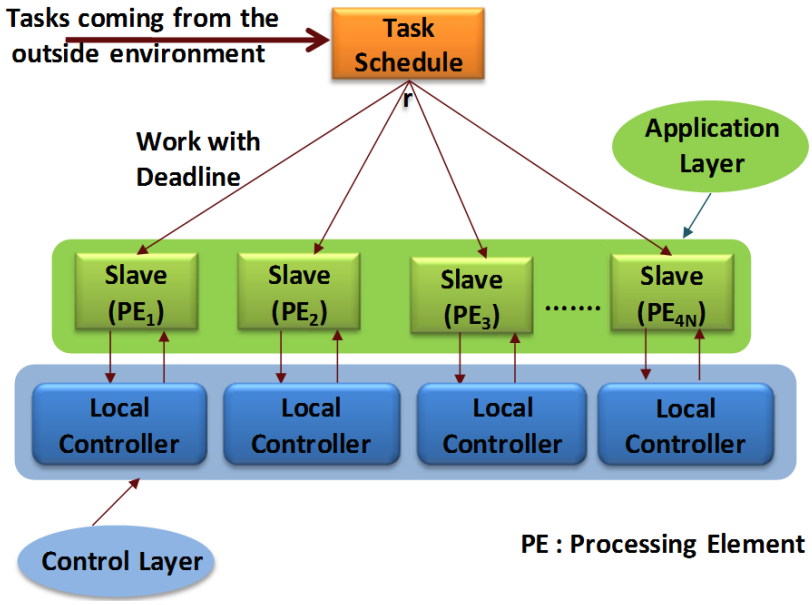

Figure A.3: The proposed two-level approach

each sample time. This approach tries to optimize a multi-objective utility function that contains conflicting parameters of execution time and power consumption.

The proposed approach (see Figure A.3) contains two levels that function independently. The top level assigns the incoming tasks (loop iterations) with deadlines in optimal numbers (equal in case of static scheduling) to the group of processors at the bottom-level. The bottom-level ensures that the best response time of each processor is achieved with minimum power consumption by monitoring the performance of individual processors even in the presence of perturbations in CPU availability. This level optimizes the loop execution, as well as the HPC system performance, through balancing the need between using the minimum power consumption and a chosen average response time. The bottom level consists of two layers: an Application Layer and a Control Layer. The application layer contains several (usually in multiples of 4) independent processing elements (pro-

cessors), which in this case execute the assigned chunk of loop iterations according to the CPU availability. The control layer contains the local controllers, which assign the control input (frequency) to the processing elements of the application layer. The control layer receives performance specification for the scientific applications in terms of recommended deadlines from the top-level, and attempts to achieve this objective with minimum power consumption. Each of the processors in the group of processors interacts with its local controller for exchanging the performance data and the optimal value of CPU frequency.

The proposed control framework consists of following key components. A System Model component describes the dynamics of the active state processing element. System model will be developed through extensive regression over the system with different possible values of the control inputs as well as environmental inputs. The most simple equation of the system dynamics can be described as:

$$x(t+1) = \phi(x(t), u(t), \omega(t)) \tag{A.1}$$

where $x(t)$ is the system state at time t, the set of user controlled system inputs is $u(t)$ (CPU frequency at time t), and $\omega(t)$ is the environment input at time t (the percentage of the CPU available to the loop execution environment). The number of loop iterations finished at time t are shown as $l(t)$ and the number of loop iterations remaining at time t are shown as $L(t)$. Here $x(t) \subset R^n$ and $u(t) \subset R^m$, where R^n and R^m represent the set of system states and control input respectively.

$$l(t+1) = \frac{\omega(t+1)}{100} * \frac{\alpha(t+1)}{\hat{W}_f} * T \tag{A.2}$$

$\alpha(t)$ is the scaling factor defined as $u(t)/u_{max}$, where $u(t) \in U$ is the frequency at time t (U is the finite set of all possible frequencies that the system can take), u_{max} is the maximum supported frequency of the processor.

$\hat{W}_f$ is predicted average service time (work factor in units of time) required to execute a single loop iteration at max frequency u_{max}.

T is the sampling time of the system.

System state *x(t)* at time *t* can be defined as the set of loop iterations executed at current time *l(t)* and the remaining number of loop iterations *L(t)*

x(t) = [l(t), L(t)]

E(t) is the power consumed by the processor at current frequency $u(t)$.

Theoretically, the power consumed by a processor is proportional to the applied frequency and square of the supplied voltage across it, while the experiment shows that this relationship is linear [39]. However, we can only monitor the overall power consumed inside a complete system through wattmeter, which accommodate the power consumed by all the devices present in the system (e.g. CPU, Memory, Hard Disk, CD-Rom, CPU cooling Fan etc.) Therefore, the measured power consumption shows a non-linear relationship with the CPU frequency, and utilization. As a result, a look-up table with near neighbor interpolation was found to be the best fit for aggregating the power consumption model of the physical machine. A power consumption model is generated by using multiple CPU

frequencies, CPU utilization values, and corresponding power consumption values similar to the ones described earlier. For the loop execution systems in the proposed approach, the model utilize the maximum available CPU that results into 100% CPU utilization at all times. Therefore, the power calculation model is only dependent upon the frequency in this case because CPU utilization is constant to 100%.

As a limitation of the current online framework, the estimation of the environmental inputs (available CPU to the loop execution process) and corresponding outputs of the system are crucial for the accuracy of the model. An autoregressive moving average model is used as estimator of the environmental inputs as per Equation A.4 and A.5. Actually, the forecasting method calculates the available CPU for the loop execution environment indirectly, by calculating the CPU utilized by other applications executing on the processing node. If other applications utilize CPU percentage equal to $\sigma(t)$ at time t, then the available CPU to loop execution will be $100 - \sigma(t)$, which will be considered as $\omega(t)$.

$$\omega(t) = 100 - \sigma(t), \tag{A.3}$$

$$\sigma(t+1) = \beta * \sigma(t) + (1-\beta) * \sigma_{avg}, \tag{A.4}$$

where β is the weight on the available CPU utilization in previous sampling time. A high value of β pushes the estimate towards current CPU utilization by other applications. A low value of β shows biasing towards average CPU utilization in past history window by the other applications.

Instead of using static β, an adaptive estimator can be used for better estimation.

$$\delta(t) = \gamma * \delta + (1 - \gamma) * |\sigma(t-1) - \sigma(t)|, \tag{A.5}$$

where $\delta(t)$ denotes the error between observed and estimated available CPU at time t, δ denotes the mean error over a certain history period and γ is determined by the experiments.

$$\beta(t) = 1 - \delta(t)/\delta_{max}, \tag{A.6}$$

where δ_{max} denotes the maximum error observed over a certain history period.

During any time interval t, the controller on processor P should be able to calculate the optimal value of the frequency f for the time interval from t to $(t + 1)$, such that the cost function $J(t+1)$ can be minimized. The desired cost function $J(t+1)$ is the conjunction of drift from the desired set point *(xs)* and power consumption *E(t+1)* with different relative weights to them. Here, *(xs)* indicates the expected number of loop iterations executed by the processor for time interval a *t* to finish the execution of all of the assigned iterates ($\frac{N}{P}$) before the deadline T_d. The controller keeps updating the *xs* based upon the total number of remaining loop iterations to be executed by the processor.

$xs = [l*, L*]$, where $l*$ is the optimal number of loop iterations desired for execution in the given time interval t, and $L*$ is the optimal number of loop iterations remaining for execution at time t, in order to finish the execution by the deadline T_d

$x(t) =$ *[l(t), L(t)]* is system state at current time

$$J(x(t+1), u(t+1)) = Q * (x(t+1) - xs)^2 + R * E(u(t+1))^2, \tag{A.7}$$

where Q and R are user specified relative weights for the drift from the optimal number of executed loop iterations *xs*, and power consumption, respectively.

The optimization problem from the controller can be described as minimizing the total cost of operating the system *J*, in a look-ahead prediction horizon using *t = 1,2, 3,...H* steps.

Finally, the chosen control input is:

$$u(t_0) = \arg\min_{u(t)\in U}(\sum_{t=t_0+1}^{t=t_0+H} J(x(t), u(t))) \tag{A.8}$$

After calculating the control input $u(t_0)$, it will be assigned to the CPU for the next time interval.

A.4 Simulations and Analysis

A simulation of the proposed approach is performed in a MATLAB R2010 simulation environment on a 3.0 GHz machine with 3 GB of RAM. We have used four CPUs to run the parallel loop execution program, and the available CPU frequencies are (1.0, 1.2, 1.4, 1.7, 2.0) in GHz. The sample time (T) for observation during the simulation is considered to be equal to 30 seconds, while the work factor (W_f) of the individual loop iterate is considered be a constant, $2X10^{-4}$ seconds. The simulations are performed with and without the perturbations due to other applications running on the same processing nodes.

The synthetic graphs indicating the CPU utilization by the overhead applications on the four computing nodes are generated with the help of a random function in MATLAB and plotted in Figure A.5 (sub-figure 4 - tagged " OtherappUtilization Statistics"). In addition, the look ahead horizon (H) for the current simulation is kept equal to a value of "2" to keep the computation overhead low. The total number of loop iterations to be executed on four processors are equal to 10^8, and the deadline for the execution is varied between 200 to 800 samples (1 sample = T seconds) depending upon the experiment settings as described in the following subsections.

A series of experiments are performed to address the deadline violation issues in case of static scheduling of loop iterations over a group of four processors with perturbations related to the CPU availability in the system. With static scheduling, each processor receives an equal amount of loop iterations and proceeds executing until it finishes its entire assignment. Experiments with this proposed framework are performed with various types of perturbations at different processors, and the results demonstrate whether the QoS objectives (deadline and power consumption) have or not been achieved. Various experiments, their settings, and their observations are described as follows.

For simulation without perturbations, 10^8 loop iterations are executed by a group of four processors before a given deadline (500 samples of 30 seconds each = 15000 seconds). No other application is utilizing the CPU on the processing nodes, which means that the CPU is fully available (dedicated) to the loop execution environment. All of the CPUs are assigned their average frequency (1.4 GHz.) from the frequency range supported by the system. The results of this experiment are shown in Figure A.4 under the tag "No

Disturbance". The results from only one processor are plotted, because all the processors have similar settings, therefore producing the same results. The results also indicate that the execution of loop iterations gets completed before the deadline of 500 samples, and that load balancing is achieved.

For simulation with perturbations, this experiment is performed in similar settings as the previous one, with the addition of CPU perturbations at each processor due to another local OS application running at each node. Plots of perturbations at four processors are shown in Figure A.5 (sub-figure 4 - tagged "OtherappUtilization Statistics"), while results of the experiments are shown in Figure A.4 under the tag "With Disturbance". The primary observation in this case is that the presence of perturbations in computational resources (CPU) results in failures to achieve the deadline (500 samples), compared to the case of no disturbance. In addition, severe load imbalance issues can be observed, as all the four processors finish their execution at different times. To address these issues, a monitoring and reconfiguring approach is needed that can reallocate the computational resources when required to meet the deadline.

For simulation with perturbations and controller, this experiment is performed in similar settings as the previous ones, in the presence of perturbations in CPU resource availability to the application and the proposed framework. In this experiment, once the proposed framework is deployed, it monitors the progress of loop execution on the CPUs, and reassigns the optimal frequency that leads the processors to achieve the deadline of executing loop iterations while keeping the power consumption low. The results of this experiment are shown in Figure A.5. According to the results, it is clear that after the deployment using

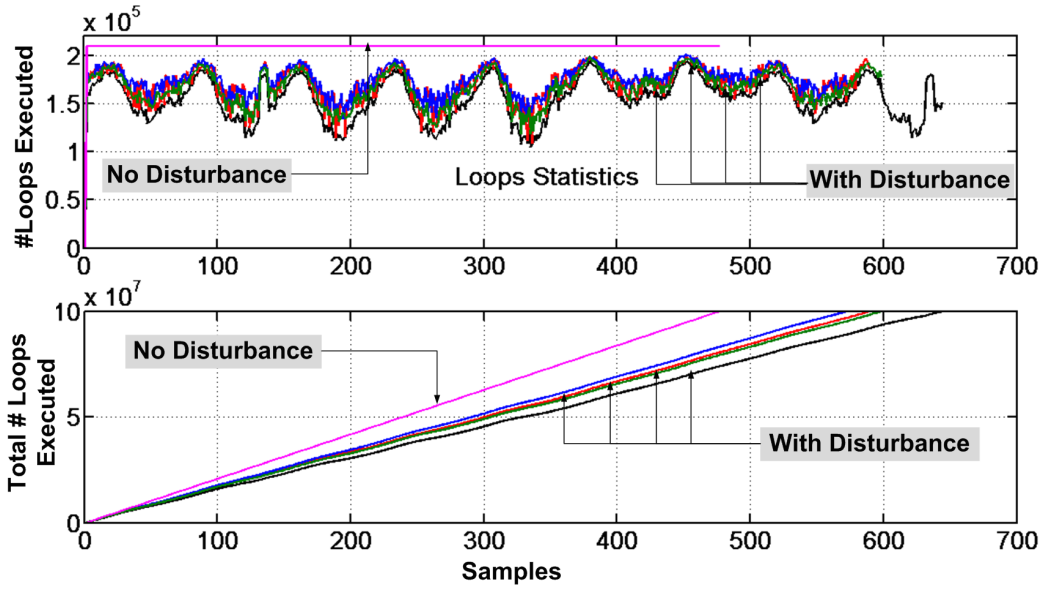

Figure A.4: Experiments performed with and without perturbation in CPU availability with deadline = 500 samples.

the proposed framework, even with the same amount of perturbations as in the previous experiment, the deadlines can easily be met by re-assigning the computational resources through changing the CPU frequency. Moreover, the results indicate considerable power savings because the CPU is not always running at its peak frequency, compared to the scenario in which the CPU is left running at its peak frequency while ignoring the perturbations inside the system.

For simulation with various priorities to deadline and power consumption, this experiment is performed to show the capability of the proposed approach in giving relative priorities to the deadline (800 samples) achievement and power consumption as described in section A.3. Three sets of experiments (deadline:power consumption - 1:1, 2:1, 4:1) are performed to show the variation in the results due to the use of relative priorities (see Figure A.6). In the case of 1:1 priority, the controller selects the optimal frequency too

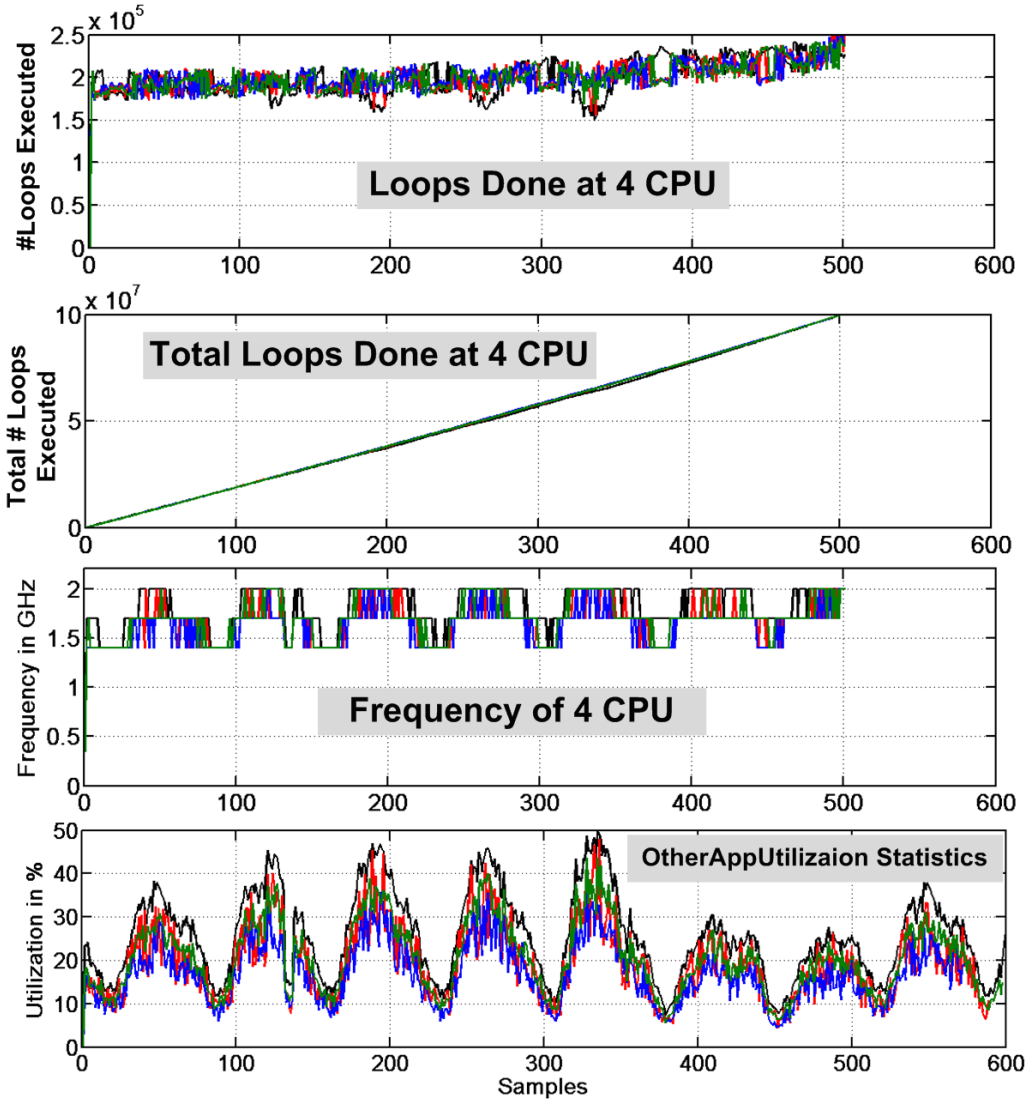

Figure A.5: Experiments performed with the proposed approach and perturbation in CPU availability with deadline = 500 samples.

conservatively (by keeping it at a minimum supported frequency value of 1.0 GHz), due to the high power consumption at higher frequencies, which in turn results in missing of the execution deadline. Even after changing the priority to 2:1, the controller continues to still conservatively select the optimal frequency, and tries to assign the higher frequencies at the end of the experiment, which in turn results in missing the deadline by a slight margin compared to the case when the relative priority used was 1:1. Finally, when the ratio is changed to 4:1, the controller starts giving priority to the deadline and changes the frequency often with respect to the perturbations in the system, which in turn results in meeting the execution deadline with efficient use of power.

In other experiments, additional simulations with various deadline values and various degree of perturbations in the system were performed. However, the results are not presented herein due to space constraints. These simulation results indicate that the proposed approach accommodate harder (400 samples) or unrealistic (200 samples) deadlines. In both of these cases, either the proposed approach achieves the deadline by assigning higher CPU frequencies (in case of harder deadlines), or assigns highest frequency in the case of unrealistic deadlines (200 samples) until the CPU finishes the execution of loop iterations.

A.5 Benefits of the Proposed Approach

The proposed approach is using a state-of-the-art methodology for performance optimization: a model-based control theoretic approach. This approach is transparent to the user and to the application. The controller and application are running as independent entities. The controller is used to tune the performance of the application while ensuring

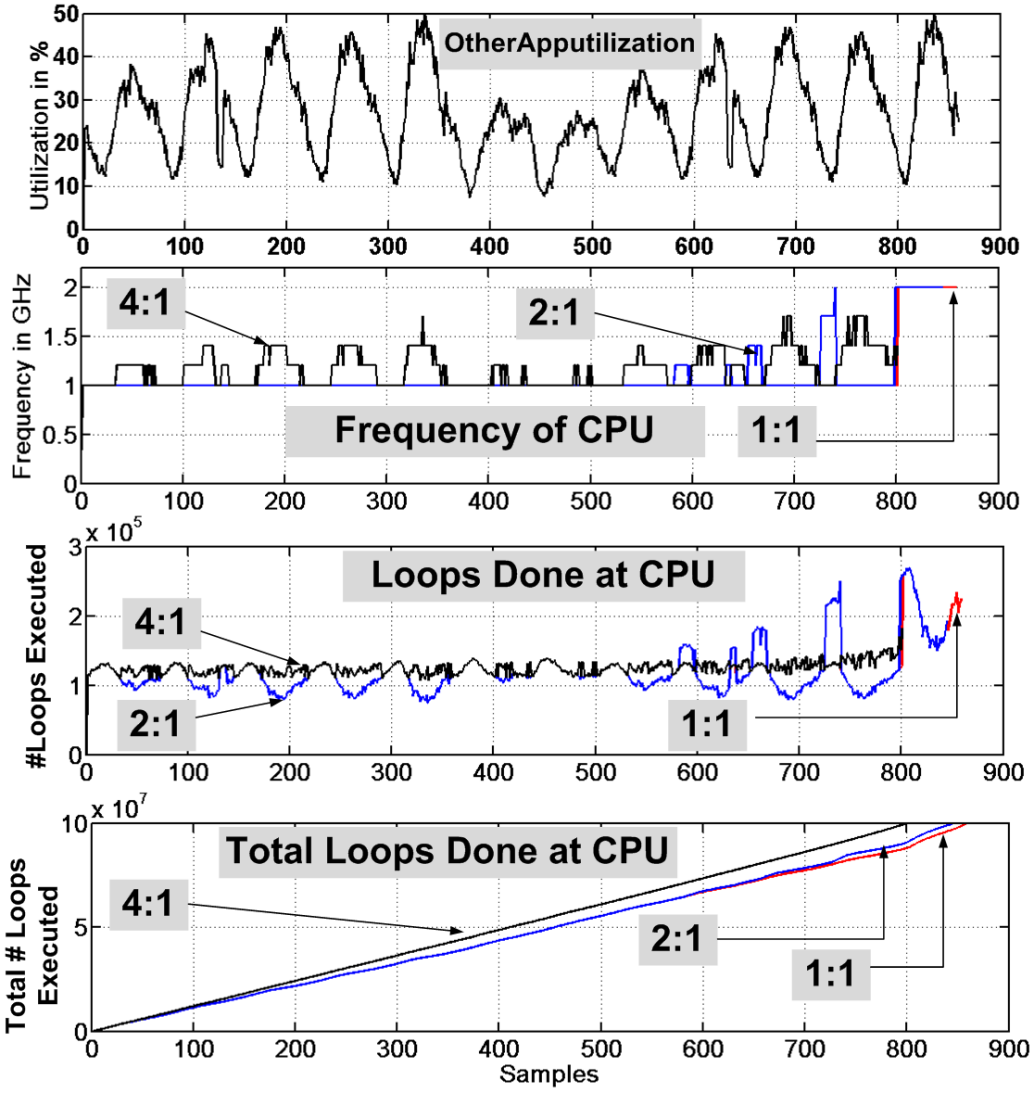

Figure A.6: Experiments performed to show the impact of relative weights to deadline and power consumption with the proposed approach and perturbation in CPU availability with deadline = 800 samples.

efficient power consumption for the overall system. The controller must have the capability to dynamically obtain measurements of the performance data while the application is running. The proposed approach is well-suited for executing scientific applications of high complexity that are scheduled in heterogeneous environments and suffer from performance degradation due to computational resource perturbations on the nodes. In addition, no code profiling or modification is needed except collecting the application performance data. The proposed framework allocates the optimal amount of computational resources for minimum power consumption in the HPC system, while keeping the deadline of the execution in consideration with the supplied priorities. Moreover, this approach provides adequate load balancing among processors.

The current approach has a single limitation, that it can only be applied to the HPC systems containing computing nodes capable of being adjusted by using DVFS techniques. Due to this limitation, the approach cannot be applied to older HPC systems, which do not have this capability. For HPC clusters, this approach is a trade-off between response time and power consumption, and can be considered as one of the solutions for achieving multi-dimensional objectives.

A.6 Conclusion and Future Work

In this paper, a model-based control theoretic and power-aware approach is presented using loop scheduling for performance optimization of HPC systems when executing scientific applications. This approach is well-suited for scientific applications of high complexity with deadline requirements. The HPC systems consume minimum power while

maintaining the predefined QoS objective of deadline (response time) with load balancing among the processing nodes. This approach provides options to the service providers to select the optimal trade-off between response time and power consumption for their infrastructure. In the future, as an extension of this work, dynamic loop scheduling methods can also be applied in conjunction with the proposed control framework for optimizing the proposed approach.